Université de Montréal

Étude de l'analyse formelle dans les données relationnelles Application à la restructuration des modèles structuraux UML

par
Mohamed Rouane Hacene

Département d'informatique et recherche opérationnelle
Faculté des arts et des sciences

Thèse présentée à la faculté des études supérieures
En vue de l'obtention du grade de philosophiæ Doctor (Ph.D.)
En informatique

Décembre, 2006

Université de Montréal
Faculté des études supérieures

Cette thèse intitulée :

Étude de l'analyse formelle dans les données relationnelles
Application à la restructuration des modèles structuraux UML

présentée par :

Mohamed Rouane Hacene

A été évaluée par un jury composé des personnes suivantes :

Jian-Yun Nie,	président -rapporteur
Petko Valtchev,	directeur de recherche
Houari Sahraoui,	codirecteur
Gena Hahn,	membre du jury
Amedeo Napoli,	examinateur externe
John Mullins,	représentant du doyen de la FES

Thèse acceptée le: 13 Décembre, 2006

Résumé

L'analyse formelle de concepts (AFC) est un paradigme de découverte de connaissances à partir des tableaux de données individus x attributs, nommés contextes, dont la structure cible, le treillis de concepts, comporte tous les groupes d'objets partageant au moins un attribut. En génie logiciel, l'AFC a été utilisé avec succès à des fins d'analyse et de restructuration de hiérarchies de classes orientées objets corrigeant ainsi des anomalies dans la répartition des membres de ces classes. Cependant, l'analyse de modèles structuraux UML réalistes s'avère hors la portée de l'approche par treillis, essentiellement à cause des aspects relationnels inhérents aux données issues de tels modèles.

La contribution de notre travail est de réconcilier l'AFC et les formats de donnés relationnelles ainsi que d'étudier l'adéquation du cadre résultant, l'analyse relationnelle de concepts (ARC), pour l'analyse de modèles UML.

L'ARC admet plusieurs contextes, chacun décrivant un type spécifique d'individus alors que les relations binaires regroupant les liens inter-individus entre les contextes sont organisées de façon séparée. Le processus d'analyse sous-jacent extrait plusieurs treillis, un par contexte, de façon simultanée et itérative. Il s'appuie sur un mécanisme de scaling qui, étant donné une relation entre deux contextes, traduit la structure conceptuelle du contexte co-domaine de la relation en un ensemble de prédicats sur les individus du contexte domaine de cette relation. Ainsi, permet-il de ramener les liens entre objets au niveau des concepts des deux contextes. La méthode concrète de construction de concepts procède par des enrichissements successifs des descriptions des individus des divers contextes, en partant des structures

conceptuelles obtenues sans les relations et en incorporant progressivement les informations relationnelles obtenues par le scaling. Ainsi, les éventuels problèmes lié à la circularité dans les liens entre individus/contextes sont résolus et le processus converge, nous l'avons démontré, vers une collection de treillis mettant en évidence tous les cas de partage de propriétés entre individus, aussi bien des propriétés d'origine que de celles obtenues par scaling, reflétant quant à elles, un partage distant, c'est-à-dire, partage par des individus reliés par des suites identiques de liens.

Vis-à-vis la problématique de réorganisation des modèles structuraux UML, nous nous sommes appuyé sur le méta-modèle UML dans la traduction d'un modèle de classes vers le paradigme de données de l'ARC. Ici, les contextes sont les types d'éléments du modèle UML et les relations sont les types de liens entre ces éléments dans le méta-modèle. Des techniques de traitement de la langue naturelle et de recherche d'information sont utilisées pour atténuer les problèmes dus à l'ambiguïté dans la sémantique des éléments du modèle. L'ARC extrait toutes les abstractions potentiellement intéressantes dans le modèle. La pertinence des abstractions par rapport au domaine est utilisée par la suite afin d'éliminer le plus grand nombre d'entre elles qui n'apportent rien à la modélisation. Finalement, un modèle UML est généré possédant une factorisation optimale des spécifications au sein de sa hiérarchie de classes.

Mots clés : Analyse formelle de concepts, théorie des treillis, modélisation de données. UML, restructuration des modèles orientés objet.

Abstract

Formal concept analysis (FCA) is mathematical method that turns a set of individuals described by properties, called *formal context*, into a hierarchy of clusters of individuals that is a complete lattice. The lattice, the set of clusters provided with a specialization order, emphasizes commonalities in descriptions (by property sets). In software engineering, a large number of studies have explored the benefits of applying FCA tools and methods in improving the structure of class hierarchies (refactoring), mainly because of the underlying simplicity and strong mathematical strength. However, when realistic structural UML models are considered, the rich knowledge encoded in the model exceeds the computational power of classical FCA, in particular because of the relational nature of some UML elements such as class associations.

The contribution of our work consists in reconciling FCA and the relational data formats as well as studying the adequacy of the resulting framework, the relational concept analysis concepts (RCA), for the analysis of models UML.

RCA helps mining potentially useful abstractions from relational data, e.g., one described by an UML model. The basic data paradigm in RCA comprises a collection of contexts, one by sort of individuals, and set of binary relations that connect individuals from these contexts. RCA provides a process that extracts a set of lattices, one by context, in a simultaneous and iterative way. The process relies on a scaling mechanism that, given a relation between two contexts, translates the conceptual structure associated to the co-domain context of the relation in a set of predicates that describe the individuals of the domain context of this

relation. Thus, links between individuals are brought to the level of concepts from both contexts. The concrete method of concept formation operates by successive enrichments of descriptions of the individuals of the various contexts, starting from the conceptual structures obtained without considering the relations and by gradually incorporating the relational information gained by the scaling. Thus, the possible problems related to circularity in the links between individuals are solved and the process converges, as we proved, towards a collection of lattices which highlight all the cases of property sharing between individuals. These properties can be initial properties or rise from scaling to reflect a distant sharing, i.e., sharing by individuals connected by identical sequences of links.

Regarding the problem of UML models refactoring, a class model is translated into a set of formal contexts, one per model element type, connected by a set of higher order binary relations that reflect the ways elements from the UML metamodel are connected within the particular model. Natural language processing and information retrieval techniques are used to attenuate the problems caused by the ambiguity in the semantics of the model elements. The RCA extracts all the potentially useful abstractions in the model. The relevance of the obtained abstractions regarding the model domain is thereafter used in order to eliminate large number of those which are not meaningful to modeling. Finally, UML model is generated offering an optimal factorization of the specifications in its class hierarchy.

Keywords : lattice theory, formal concept analysis, data modelling, UML, object oriented system refactoring.

Table des matières

Liste des tableaux

Table des figures

Remerciements

Efficacité, compétence, disponibilité, rigueur et ouverture d'esprit sont là quelques unes des nombreuses qualités que j'ai trouvé en mon directeur de recherche. Je le remercie sincèrement pour m'avoir appuyée scientifiquement et financièrement. Parfois mettant du stress, parfois lâchant du lest, le travail devait avancer régulièrement. Je remercie mon codirecteur pour m'avoir soutenu et d'avoir répondu à mes nombreuses questions sur le génie logiciel. La pertinence de ses réponses et de ses questions m'ont beaucoup appris sur mon propre sujet et sa sympathie a apporté beaucoup de détente au milieu d'un travail ardu.

Je remercie le professeur Jian-Yun Nie d'avoir accepté de présider ce jury. Je remercie également le professeur Gena Hahn pour avoir accepté d'être membre du jury. Je remercie spécifiquement et sincèrement le professeur Amedeo Napoli d'abord d'avoir accepté de faire partie du jury et aussi de nous avoir enrichi par tous ses commentaires très pertinents. Sa présence dans le jury m'honore.

Finalement, que tous ceux et celles qui m'ont encouragé et fait des invocations pour moi, parmi mes chers parents, mes chers frères et soeurs, parmi ceux de ma grande famille, tous mes amis où qu'ils soient, trouvent ici autant et même plus de gratitude et profonde sympathie que leur soutien et encouragements. Mes remerciements vont aussi à tous les professeurs et collègues du laboratoires GEODES qui m'ont gentiment accepté et aidé. Un tel travail, de longue haleine, n'a pu aboutir que grâce à l'implication, d'une manière ou d'une autre, de toutes ces personnes.

Chapitre 1

Introduction

1.1 Contexte de la recherche

Le travail s'inscrit dans le cadre du traitement de données complexes en intelligence artificielle et de la découverte de connaissances à l'aide des treillis de concepts. Le but est de fournir des outils effectifs d'analyse de données et métadonnées complexes et plus particulièrement, de supporter les restructurations de modèles conceptuels, exprimés en UML. Ainsi, nous étudions la problématique de l'introduction d'aspects relationnels dans l'analyse formelle de concepts, en suivant les répercussions de l'approche proposée sur les constructions fondamentales, les mécanismes algorithmiques et les outils logiciels, jusqu'à l'application de ceux-ci dans le contexte de la restructuration de modèles conceptuels orientés objets en génie logiciel.

1.2 Motivation

Nous travaillons sur le langage de modélisation UML [27, 42, 66] qui fournit une notation standard pour la description des modèles à objet en génie logiciel. UML permet l'expression, l'organisation et la communication de la connaissance sur un domaine particulier en s'appuyant sur différents artéfacts notamment le

diagramme de classes qui reflète les caractéristiques structurelles des entités du monde réel, leurs abstractions et leurs relations en terme de classes et d'associations. La Figure 1.1 illustre un diagramme de classes qui représente une partie du modèle d'un système imaginaire de gestion et d'organisation du temps tel que l'outil iCal du système d'exploitation Mac OS X. Les classes `Time` (temps) et `Date` dénotent un point dans le temps et un jour de l'année, respectivement, alors que les classes `Diary` (agenda) et `Clock` (Horloge) désignent des indicateurs de dates et d'heures, respectivement.

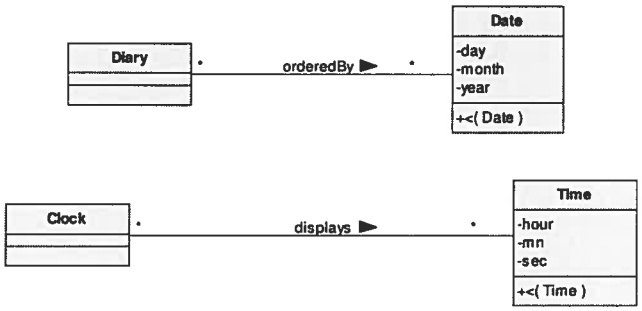


FIG. 1.1 – Un diagramme de classes en UML.

La richesse d'expression de UML est accompagnée d'une relative difficulté de construction du modèle [26, 52]. Cette difficulté est dûe, d'une part, au fait que UML est une notation et non pas une méthode permettant la construction des modèles d'une manière systématique ; et d'autre part, à la complexité que présente l'activité de modélisation, notamment un espace de recherche très large, des critères parfois conflictuels et une évolution permanente en rapport avec la réalité [9]. Par conséquent, en modélisation UML, il faut s'assurer que les artéfacts vérifient tous les critères de qualité [26, 52] notamment la présence des concepts pertinents du domaine modélisé et la factorisation maximale des spécifications, voulant que des éléments du modèle ne se trouvent pas définis à plusieurs endroits[1]. La factorisation maximale permet une réduction de l'effort de maintenance du modèle et son exten-

[1]Dans plusieurs classes du diagramme

sibilité par un meilleur partage des spécifications. Cependant, dans les modèles de grande taille et les modèles qui subissent des évolutions, il arrive souvent que la factorisation maximale n'est pas respectée. Pour illustrer ce phénomène, considérons le diagramme UML de la Figure 1.1. Les deux classes `Date` et `Time` partagent la méthode $\leq$ dont la déclaration se retrouve répétée. Alors on imagine facilement qu'une nouvelle classe, appellerons-la `Datum`, pourrait être créée pour centraliser la définition de la méthode $\leq$ (voir Figure 1.3). Afin d'encadrer ce genre de réorganisation et de s'assurer que toutes les restructurations possibles d'un modèle UML ont été détectées, des méthodes automatiques ont été conçues [2, 15, 45]. Le courant dominant dans ce champ est constitué par les approches basées sur l'analyse formelle de concepts (AFC) [32, 35, 41, 77]. En effet, un modèle structurel UML peut être vu comme étant composé d'un squelette, la hiérarchie de classes, sur lequel viennent se greffer les autres éléments du modèle tels que les attributs, méthodes, associations, etc. L'héritage fait de sorte qu'une classe représente un regroupement de ses sous-classes qui partagent les éléments définis ou hérités par cette classe. Par conséquent, les approches de classification et de structuration conceptuelle telle que l'AFC peuvent être utilisées de façon avantageuse dans la construction et/ou la restructuration[2].

L'AFC [30, 95] est une approche pour la découverte et la structuration de connaissances. Elle fournit une méthode universelle 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*) faisant apparaître un ensemble d'individus (*objets formels*[3]), un ensemble de caractéristiques (*attributs formels*) et exprimant la relation d'incidence objet-attribut. Par exemple, la Table 1.1 montre un contexte binaire où les objets formels sont les classes du modèle UML illustré par la Figure 1.1 et les attributs formels sont les variables et les méthodes des différentes classes.

Les algorithmes de l'AFC permettent l'extraction de tous les groupes d'individus

[2]Nous utiliserons le mot 'restructuration' pour désigner le mot Anglais 'refactoring'.

[3]Dans cette thèse, les expressions 'individu' et 'objet formel' ont le même sens.

	day	month	year	hour	mn	sec	≤ (Date)	isLeapYear()	≤ (Time)	≤ (Object)
Date	X	X	X				X	X		X
Time				X	X	X			X	X

TAB. 1.1 – Codage des classes du modèle UML de la Figure 1.1 en un contexte binaire.

ayant des attributs en commun. Ces groupes, appelés *concepts formels*, sont décrits par les attributs partagés et organisés en une hiérarchie, appelée *treillis de concepts*. Le groupe d'individus et le groupe d'attributs formant un concept sont appelés *extension* et *intension*, respectivement. Par exemple, la Figure 1.2 montre le treillis de concepts obtenu à partir du contexte de la Table 1.1. Il est à remarquer que le concept formel 0 est composé de l'extension $\{Date, Time\}$ et de l'intension $\{\leq (Object)\}$. Cela signifie que les deux classes `Date` et `Time` partagent la méthode $\leq$ qui a un objet générique comme paramètre.

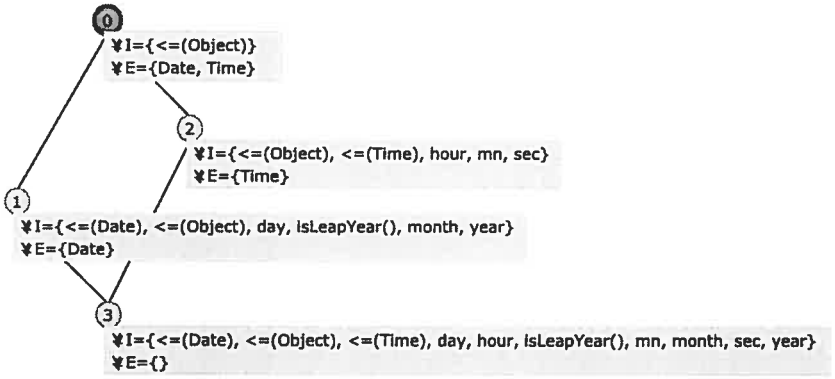


FIG. 1.2 – Le treillis de concept qui correspond au contexte de la Table 1.1.

Les méthodes d'analyse de l'AFC sont basées sur la construction, la visualisation et l'exploration du treillis ou de ses structures dérivées. Vis-à-vis des difficultés dans la création et la maintenance du modèle structurel UML, ces méthodes offrent un

environnement complet de représentation et de découverte de connaissances sur la structure des éléments du modèle grâce aux propriétés intrinsèques d'un treillis de concepts telles que :

- **Maximalité :** les extensions/intensions des concepts représentent des groupes maximaux dans la mesure où un concept est l'ensemble maximal d'individus partageant un ensemble maximal de caractéristiques. Cette notion favorise la factorisation et diminue le nombre de groupes d'individus à créer.
- **Exhaustivité :** le treillis représente un espace de recherche exhaustive pour les méthodes de construction de classifications car tous les groupements significatifs (concepts) sont présents. En conséquence, toutes les hiérarchies de concepts, c'est à dire, les solutions potentielles, y sont présentes et il s'agit de détecter la ou les solutions adéquates.
- **Factorisation maximale :** au sein d'un treillis, il existe un concept maximal pour toute propriété d'individu, appelé *concept-attribut*. Tout autre concept ayant la même propriété est inférieur au concept-attribut. Lorsque on traduit en classes le treillis de concepts, cela entraine l'existance d'une seule classe qui définit la propriété et dont toutes les sous-classes l'héritent.

A ces propriétés 'statiques', on pourrait rajouter le fait qu'en tant que méthode de regroupement, l'AFC fournit un résultat qui ne dépend pas de l'algorithme concret employé. De plus, elle fournit un cadre formel et par là, des méthodes exactes pour d'autres opérations de granularité plus fine (par exemple, rajout/suppression d'objets ou attributs) ou bien plus élevée (fusion/scission de hiérarchies conceptuelles) qui ont leur importance pour de nombreuses problématiques en génie logiciel telles que la maintenance des modèles et l'intégration de modèles partiels.

Bien que les techniques de l'AFC ont été appliquées avec succès en génie logiciel, la richesse des données rencontrées dans ces applications, en fort contraste avec la relative simplicité du modèle binaire de données utilisé en AFC, a révélé les carences du paradigme '*classique*' de l'AFC. En effet, le treillis de la Figure 1.2 par exemple, nous a permis de détecter une seule abstraction possible dans le modèle UML de la Figure 1.1, à savoir la classe `Datum` généralisant `Date` et `Time` alors

que d'autres auraient été possibles. Par exemple, les deux classes `Clock` et `Diary` jouent un rôle analogue par rapport aux classes `Date` et `Time`. Intuitivement, ces classes pourront être généralisées par une super-classe commune qu'on pourrait appeler `DatumIndicator` et qui factoriserait leur rôle d'indicateur d'une donnée. En se penchant sur les éléments du modèle qui pourraient servir de base pour cette dernière abstraction, on constate que les deux classes `Diary` et `Clock` sont incidentes à des associations à sémantique proche les reliant à des sous-classes de `Datum`. Donc, deux questions surviennent naturellement : la première est comment reconnaître le fait que les deux associations ont une sémantique 'proche' ? et la seconde est comment représenter la généralisation de `Date` et `Time` en `Datum` vers `Clock` et `Diary` ?. Alors que la première question appelle clairement un traitement linguistique (par exemple, une analyse des sens des deux termes), la seconde sort clairement du cadre de l'AFC et requiet de nouveaux mécanismes d'abstraction. Ces mécanismes devraient permettre de regrouper les objets formels, en l'occurrence, les classes `Clock` et `Diary` en fonction de ce qu'il y a en commun non seulement dans leurs propres descriptions, mais aussi dans les descriptions des objets qui leurs sont liés (`Date` et `Time`).

	isDescribedBy	has	isOrigin	isDestination
Diary			{orderedBy}	
Clock			{displays}	
Date	{day,month,year}	{≤(Date),isLeapYear}		{orderedBy}
Time	{hour,mn,sec}	{≤(Time)}		{displays}

TAB. 1.2 – Codage des liens entre les éléments du modèle UML de la Figure 1.1.

Pour aller encore plus loin avec UML, nous aimerions pouvoir creuser davantage dans le modèle pour abstraire non seulement un seul type d'élément de modélisation tel que les classes mais n'importe quel autre type d'élément du modèle tel que les attributs de classes, méthodes, rôles, associations, etc. Cependant, ces types d'éléments sont reliés entre eux par des relations telles que l'incidence classe-attribut, le typage d'un rôle par une classe, etc. La Table 1.2 montre quelques liens entre les classes et leurs attributs, méthodes et associations du modèle UML

de la Figure 1.1. Ces liens constituent des informations qu'on pourrait qualifier de *relationnelles.*

L'obstacle devant les méthodes automatiques capables de produire des abstractions (voir l'association `indicates` et la classe `DatumIndicator` sur la Figure 1.3) est de taille car il s'agit de modifier la façon dont les concepts sont construits en y intégrant le partage de liens à coté de celui des attributs.

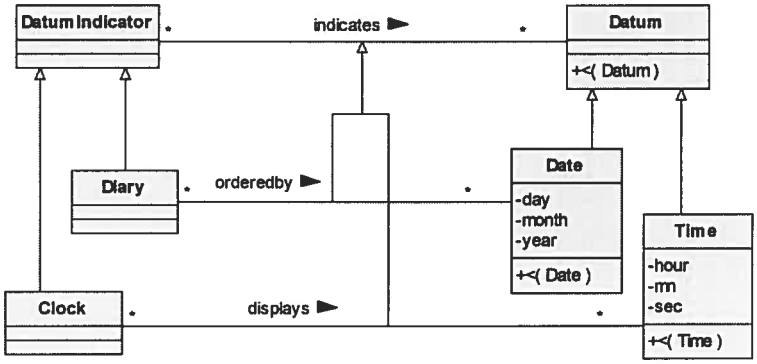


FIG. 1.3 – Le diagramme de classes de la Figure 1.3 après restructuration.

1.3 Problématique

Le besoin d'extraire des abstractions potentiellement intéressantes des modèles structuraux UML réalistes nous a conduit à l'incorporation des informations relationnelles propres à ces modèles, telles les incidences entre classes et associations, dans le processus même de construction de ces abstractions. La problématique sous-jacente n'est pas spécifique au traitement de données ou, comme c'est le cas ici, de méta-données reliées à la modélisation en génie logiciel. Des informations relationnelles peuvent être rencontrées dans un grand nombre de situations relevant de l'analyse de données telle que la génétique avec, entre autre, l'analyse des structures 2- et 3-dimensionnelles des macro-molécules de l'ADN (les liens ou ponts entre nucléotydes) ou l'étude des interactions entre gènes et protéines. De

façon plus générique, on retrouve des informations relationnelles dans toute base de données relationnelle, par exemple, codées sous la forme de clés étrangères dans une table de la base. Dans toute sa généralité, la problématique que nous étudions ici se range sous la bannière de la fouille de données relationnelles qui constitue un champs particulièrement prolifique au sein de la fouille de données et la découverte de connaissances dans les bases de données [1, 22].

Comme il a été souligné plus haut, les relations ou liens inter-objets ne s'accordent pas facilement avec le modèle de données classique en AFC, à savoir les collections de propriétés, ni d'ailleurs avec les formats plus avancés incorporant des attributs valués. Même si des tentatives de réconciliation de l'outillage de construction de concepts formels et des représentations plus expressives on été entreprises, celles-ci ne nous permettent pas de résoudre le problème identifié dans sa globalité car les formalismes résultants, bien qu'admettant des liens entre entités, renferment ceux-ci au sein des objets formels ou bien ils ne considèrent pas les liens inter-objets lors de la construction des concepts. Ce constat vient justifier une approche globale vers l'analyse formelle de données relationnelles qui va jusqu'à redéfinir les éléments clés de la discipline car l'introduction de nouveaux formats pour les données risque d'avoir un impact sur tout autre aspect du processus d'analyse.

En effet, une étude comme la notre doit aborder un certain nombre de questions relevant de toute tentative de définir un type de traitement analytique sur des données pour lesquelles celui-ci n'a pas été prévu à l'origine. Parmi celles-ci le format des données en entrée n'est que la première préoccupation. En réalité, deux langages doivent être conçus, l'un décrivant les entités à analyser et l'autre, les résultats, en l'occurrence, les concepts formels. Étant donné que notre objectif est d'incorporer des aspects relationnels dans la description des individus, les nombreux formalismes de représentation de connaissances structurées tels que les logiques de descriptions ou les graphes conceptuels auraient pu, a priori, servir pour nos deux langages. Même UML aurait pu être choisi dans le rôle de langage de description, surtout que le langage dispose depuis peu d'un méta-modèle complet. Une telle approche reviendrait à substituer, de façon plus ou moins radicale, les ensembles

d'attributs par des expressions d'un langage plus riche et entraînerait la nécessité de redéfinir les mécanismes de construction de concepts, voir, la notion de concept formel elle-même. L'approche alternative consisterait à compléter le format actuel utilisé en AFC et orienté vers le traitement d'attributs binaires avec des descripteurs reflétant les liens entre objets formels. Ceux-ci pourront par la suite être ramenés, par des techniques de scaling, vers des descriptions binaires. Bien que les descriptions de données n'auront pas toute la richesse des langages cités ci-dessus, elles ne demanderont que peu de reformulations au sein des mécanismes d'analyse classiques d'AFC. Il est à noter également que la relative simplicité du langage des données n'enfreint en rien la puissance d'expression du langage des concepts.

Un autre aspect du cadre d'analyse relationnelle esquissé qui devrait être abordé couvre la conception des mécanismes adaptés à la construction de concepts. Tout comme dans le cas binaire, le langage de description des concepts va admettre des descriptions de deux types, les intensions, c'est-à-dire les descriptions fermées, et des non-intensions. La construction de concepts pourra donc être vue comme une recherche dans l'espace de toutes les descriptions du langage des concepts et la sélection des intensions, suivi de la construction des extensions correspondantes. Sur ce point, un choix de langage de données basé sur les attributs permettra de bénéficier des définitions actuelles pour ces notions, même si certains attributs utilisés dans l'expression des information relationnelles pourront nécessiter une interprétation post-analyse car chargés d'une sémantique particulière. En revanche, la détermination des expressions fermées au sein d'un langage expressif pourrait nécessiter un effort de reformulation et/ou conception de mécanismes considérables, se heurtant, dans certain cas, à des problèmes difficiles de raisonnement automatique (par ex., le test de la subsomption dans les logiques de descriptions expressives). Ces dernières pourront influencer à leur tour les stratégies concrètes de parcours de l'espace des descriptions, comme on l'observe dans le cas de la fouille de motifs fermés fréquents à partir de données complexes (par exemple, des graphes comme dans [97]).

Les choix pris sur cet aspect se refléteront, bien entendu, sur la conception

d'algorithmes de construction de treillis concrets ainsi que sur la réalisation d'outils logiciels d'analyse relationnelle. Là encore, une approche plus traditionnelle utilisant des transformations vers des ensembles d'attributs permettra de profiter de l'ensemble de techniques algorithmiques existantes dans la littérature ou disponibles au sein des systèmes d'AFC comme ToscanaJ ou Galicia. De façon similaire, l'adoption d'un langage de concepts plus expressif, risquerait de forcer la conception d'algorithmes nouveaux de construction et navigation de treillis. De plus, l'implémentation de ceux-ci pourrait poser des problèmes supplémentaires lors de l'intégration au sein d'un outil logiciel existant, en particulier, à cause des formats des données en entrée.

Finalement, l'interprétation des résultats de l'analyse, c'est-à-dire le treillis des concepts, est un point à ne pas négliger. Ici les solutions consistant à adopter des langages riches de description pour les données et les concepts définis dans la littérature, a l'avantage d'offrir des mécanismes d'interprétation bien connus. Au contraire, la réduction des aspects relationnels à des collections d'attributs binaires nécessiterait un effort supplémentaire de la part de l'analyste lors de la phase post-AFC afin de restituer la sémantique des liens. Cet inconvénient pourrait être atténué par l'utilisation d'une traduction — automatique ou semi-automatique — des concepts du treillis vers un langage riche.

Concernant notre problématique de restructuration de modèles UML, deux aspects additionnels devraient être abordés dont l'origine se situe dans la nature des données manipulées. En effet, étant donné que les données à analyser proviennent d'un modèle, c'est-à-dire, qu'elles représentent des méta-données, il sera crucial de s'assurer que lors de l'inévitable traduction vers le format choisi pour l'entrée de notre processus d'analyse, ainsi que la traduction inverse des concepts formels retrouvés en un nouveau modèle, la sémantique du modèle initial soit respectée.

1.4 Travaux antérieurs

Notre étude se situe dans la continuité des travaux de Huchard et al. [41] qui ont abordé la restructuration des modèles UML sous un angle spécifique au génie logiciel. Les résultats de ce travail mené en collaboration entre les équipes à Montréal et à Montpellier, sont exposés de façon complète dans la thèse de doctorat de Cyril Roume [70]. Leurs contributions majeures pourront être résumés comme suit. Tout d'abord, une réponse à la question de la représentation des informations relationnelles a été apportée par l'adoption d'une structure de données incorporant plusieurs contextes, un par type d'individus à analyser, ainsi que des représentations indépendantes pour les diverses sortes de liens existant entre les individus. Des mécanismes de codage de ces liens vers des attributs binaires ont été proposés ainsi qu'un processus itératif de construction de concepts formels, ICG, palliant les problèmes de circularité dans les descriptions des objets issus d'un modèle UML. Des traductions entre le méta-modèle UML et les entités d'AFC, objets, attributs, liens, concepts, ont été définies d'une façon qui, dans la plupart des cas, reste ad hoc. L'approche pour l'analyse de modèles UML a été réalisé au sein d'un outil existant, Galicia, et a pu être validée dans le cadre d'un projet mixte université-industrie, MACAO[4].

1.5 Approche

L'objectif de notre propre travail est de consolider, c'est-à-dire, compléter et enrichir, les travaux initiaux de Huchard et al. afin de produire un cadre complet d'analyse de données relationnelles à l'aide de treillis de concepts. Ceci entraîne la systématisation des résultats structuraux déjà obtenus et la consolidation des bases théoriques, en profitant des avancées récentes dans la maîtrise des aspects relationnels en AFC.

Ainsi, nous n'avons pas remis en cause les choix initiaux quant à la représentation

[4]http ://www.lirmm.fr/~macao.

des données de l'analyse et nous avons adapté la structure sous-jacente, appelée famille de contextes relationnels. En revanche, une variété de façons de ramener les liens inter-objet vers des attributs binaires a été considérée avec les mécanismes sous-jacents de traduction. Leur formalisation a mené à l'étude de nouveaux types de scaling, dédié aux liens inter-objets, avec des politiques variables pour l'attribution des attributs synthétisés aux objets propriétaires des liens. En conséquence, l'interprétation des résultats de l'analyse a été placée dans un contexte générique dépassant les besoins spécifiques de la reconstruction d'un modèle structurel UML. Ainsi, la traduction vers d'autres formalismes de représentation, facilitant l'interprétation a été examinée, avec un accent sur les logiques de descriptions dont l'utilisation à cette fin a été jugée très pertinente. La composition des résultats de l'analyse en tant que structures partiellement ordonnées a également été abordée, en cherchant à clarifier le rapport, jusqu'alors inconnu, entre les liens inter-objets en entrée et le regroupement des objets en concepts. En profitant du formalisme, nous avons également attaqué le processus de construction lui-même, d'une part, pour étudier son comportement (terminaison, rapidité de la convergence, robustesse) et d'autre part, pour proposer des optimisations en vue d'améliorer ses performances en temps de calcul.

Finalement, concernant l'analyse d'artefacts UML, les traductions entre les modèles logiciels et les structures d'AFC ont été abordées sous l'angle de la standardisation et de la transformation de modèles (dans l'esprit de l'approche MDA[5]), avec comme objectif la définition de traductions canoniques s'attachant directement aux éléments du méta-modèle UML 2.0. De plus, nous avons considéré le problème de l'ambiguïté sémantique inhérente aux modèles UML qui empêche l'automatisation complète du processus de traduction et avons examiné des solutions de principe. Nous sommes allés jusqu'à la formalisation de la traduction inverse avec l'étude de processus dédiés à la reconstruction du modèle UML final à partir des treillis obtenus par l'analyse.

[5]http ://www.omg.org/mda/.

1.6 Contribution

Les avancées suite à nos travaux sur le sujet peuvent être résumées comme suit :

- Plan fondamental : la formulation générique du problème, l'expression analytique de la solution, l'étude des divers modes de scaling relationnel, et la preuve de convergence du processus itératif de solution.
- Plan algorithmique : pour que le cadre soit effectif, la partie algorithmique a été aussi développée et des solutions appropriées aux problèmes sous-jacents tels que le traitement des dépendances circulaires et la conception d'algorithmes incrémentaux ont été apportés en se basant sur ce qui a été proposé pour le cas de l'application aux modèles UML. Les développements sont implémentés au sein de l'outil GALICIA qui permet leur expérimentation et validation. Le cadre étendu de l'AFC proposé à travers GALICIA permet d'assister la tâche d'analyse d'une collection arbitraire d'individus pour peu que la description est compatible avec un modèle UML.
- Plan applicatif : dans une autre veine, plus orientée vers les applications en génie logiciel, nous avons défini une traduction canonique et bidirectionnelle entre un modèle structurel UML, d'une part, et une famille de contextes (FCR), d'autre part, de telle sorte que les résultats des opérations supportées par le cadre AFC étendu puissent être facilement traduits en un modèle UML. Les difficultés d'ordre linguistiques liées au codage du modèle UML en une FCR notamment celles qui émergent des conflits de noms des différents éléments du modèle ont été étudiés et des solutions ont été proposées et implémentées dans des outils. Finalement, des travaux de validation de l'approche proposée ont été conduits sur des données industrielles.

1.7 Plan de la thèse

Cette thèse est organisée en plusieurs chapitres :

Le chapitre 2 introduit le cadre fondamental de l'AFC. On y trouve les notions fondamentales des treillis de concepts et certains mécanismes incontournables en AFC tel que le scaling conceptuel et les différentes approches de construction du treillis de Galois. Le chapitre présente aussi quelques applications usuelles de l'AFC, notamment en génie logiciel. Étant donné que nos travaux s'inscrivent dans ce dernier domaine, une description de la manière avec laquelle l'AFC a abordé les problèmes de réingénierie des systèmes logiciels et les solutions proposées pour améliorer leur qualité seront présentées.

Le chapitre 3 étudie les principales extensions de l'AFC qui ont été proposées dans le but de traiter des données complexes. Il présente aussi le modèle conceptuel d'UML ainsi que les difficultés rencontrées dans la construction de celui-ci.

Le chapitre 4 présente les fondements théoriques de l'ARC. On y trouve le modèle de données de l'ARC ainsi que le mécanisme de base de traitement des liens inter-objets formels (scaling relationnel) permettant la dérivation de structures conceptuelles où les concepts peuvent dépendre d'autres concepts.

Le chapitre 5 décrit les aspects algorithmiques de l'ARC. Il présente le schéma général de la méthode de construction des treillis relationnels au-dessus d'une FCR, dite 'MULTI-FCA' ainsi que les principaux algorithmes auxilliaires, notamment l'algorithme de scaling relationnel et l'algorithme de maintenance d'un treillis par ajout d'attributs.

Le chapitre 6 est consacré à la description de la manière de coder un modèle structurel UML[6] en une FCR, le format d'entrée de l'ARC. Le chapitre aborde les conflits de noms entre les éléments du modèle lors de la production de la FCR. Le chapitre présente aussi les règles de traduction des résultats de l'analyse (une famille de treillis relationnel) en un modèle UML.

Le chapitre 7 présente notre contribution au développement de la plateforme

[6]Version 1.5 ou plus.

GALICIA. Il décrit les outils qui implémentent les méthodes et les mécanismes de l'ARC. On y trouve une présentation de la plateforme d'expérimentation GALICIA qui est le projet pilote de l'ARC ainsi que l'outil RCFMODELER permettant d'interfacer GALICIA avec les ateliers de génie logiciel. Le chapitre met l'emphase sur les méthodes de calcul de similarité dans RCFMODELER afin de lever l'ambiguïté sur les noms des éléments d'un modèle structurel UML lors de la construction de la famille de contextes.

Le dernier chapitre met en évidence l'intérêt de l'ARC dans la restructuration des modèles conceptuels de classes UML par le biais d'expérimentations effectuées sur des modèles de taille réaliste. Le chapitre présente en détail l'application de l'ARC au modèle du système JETSGO tandis que le projet MACAO est décrit très brièvement.

Chapitre 2

Analyse Formelle de Concept

Ce chapitre est une introduction au treillis de concepts d'une relation binaire (individus × propriétés) et à l'AFC. Nous présentons d'abord les notions fondamentales de cette théorie, puis nous abordons les différentes structures conceptuelles. Par la suite, nous décrivons le moyen de codage des données non binaires en AFC et le mécanisme qui s'y rattache. Nous allons aussi aborder les algorithmes de construction du treillis de concepts. Enfin du chapitre nous décrivons quelques application usuelles.

2.1 Fondements de l'AFC

Nous allons présenter les notions fondamentales de l'AFC, les structures conceptuelles qu'elle utilise et le mécanisme de scaling conceptuel qui permet de traiter les contextes non binaires comme nous allons le voir.

2.1.1 Quelques éléments de la théorie des treillis

Dans cette section, nous allons nous contenter d'exposer certaines notions fondamentales de la théorie des treillis[1] en s'inspirant de [17] et de [30].

[1]Une excellente introduction aux ordres partiels et treillis peut être trouvée dans [17]

2.1.1.1 Ensembles ordonnés

Définition 1 (ordre ou ordre partiel). *Soient E un ensemble et $\leq$ est une relation binaire sur E. Un ordre partiel sur E donné par la relation $\leq$ vérifie, pour tout x, y et $z \in E$, les propriétés suivantes :*

- *$\leq$ est réflexive : $x \leq x$,*
- *$\leq$ est antisymétrique : si $x \leq y$ et $y \leq x$ alors $x = y$,*
- *$\leq$ est transitive : si $x \leq y$ et $y \leq z$ alors $x \leq z$.*

Exemple : La relation "divise" est un ordre partiel sur l'ensemble des nombres entiers naturels $\mathbb{N}$.

Définition 2 (ensemble ordonné). *Un couple $(E, \leq)$ formé d'un ensemble E et d'une relation d'ordre $\leq$ est appelé ensemble ordonné.*

Dans un ensemble ordonné $(E, \leq)$, pour une paire d'éléments $x, y \in E$, on dira que x et y sont comparables si $x \leq y$ ou $y \leq x$. Si tous les éléments sont comparables, on dit que l'ordre est total, sinon il n'est que partiel. Pour deux éléments comparables et différents, $x \leq y$ et $x \neq y$, on note $x < y$.

Les éléments d'un ensemble ordonné sont liés par la "*relation de couverture*" induite par l'ordre. Formellement, la couverture est définie de la manière suivante :

Définition 3 (relation de couverture). *Soient $(E,\leq)$ un ensemble ordonné et $x, y \in E$. On dit que y couvre x (ou y est un successeur de x, x est un prédécesseur de y), noté $x \prec y$, si $x < y, x \leq z < y \Rightarrow z = x$.*

2.1.1.2 Diagramme de Hasse d'une relation d'ordre

Un aspect attrayant et très utile des ensembles ordonnés est la représentation graphique. En effet, tout ensemble ordonné $(E,\leq)$ peut être représenté de façon schématique par un diagramme appelé "'diagramme de Hasse" qui est un graphe où les nœuds sont les éléments de E et les arcs représentent la relation de couverture entre éléments. La réalisation du diagramme consiste à :

- Étape (i) : $\forall x \in E$, associer un point $p(x)$ dans le plan cartésien,

- Étape (ii) : $\forall x, y \in E$, tel que $x \leq y$, tracer un segment $l(x, y)$ entre $p(x)$ et $p(y)$,
- Contraintes : faire l'étape (i) et (ii) en respectant les conditions suivantes :
  - Si $x \leq y$, alors l'ordonnée de $p(x)$ est inférieure à celle de $p(y)$,
  - Un point $p(z)$ ne peut se trouver sur le segment $l(x, y)$ si $z \neq x$ et $z \neq y$.

Exemple : Soit E l'ensemble formé par les diviseurs de 30. $E = \{1, 2, 3, 5, 6, 10, 15, 30\}$ et $\leq$ la relation binaire "divise", notée $\div$. La Figure 2.1 montre le graphe associé à $\leq$.

	1	2	3	5	6	10	15	30
1	x	x	x	x	x	x	x	x
2		x			x	x		x
3			x		x		x	x
5				x		x	x	x
6					x			x
10						x		x
15							x	x
30								x

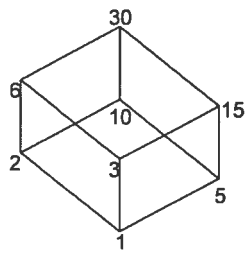


FIG. 2.1 – La relation $\div$ et le diagramme de Hasse de $(E, \div)$.

2.1.1.3 Éléments particuliers d'un ensemble ordonné

Définition 4 (Chaîne et antichaîne). *Une chaîne d'un ensemble ordonné $(E, \leq)$ est une partie S de E tel que quel que soit $(x, y) \in S^2, x \leq y$ ou $y \leq x$. Un ensemble tel que $x \leq y \Rightarrow x = y$ est appelé une antichaîne. Une chaîne S (resp. antichaîne) est dite maximale si et seulement si quel que soit l'élément $x \in E \setminus S$, l'ensemble $S \cup \{x\}$ n'est pas une chaîne (resp. antichaîne).*

Exemple : Sur la Figure 2.1, à droite, la séquence de nœuds $[1, 5, 15]$ forme une chaîne.

Définition 5 (majorant - minorant). *Soient $(E, \leq)$ un ensemble ordonné et S une partie non vide de E. On dit qu'un élément a est un majorant (respectivement un minorant) de S, si pour tout x de S, $x \leq a$ (respectivement $x \geq a$).*

L'ensemble S peut avoir plusieurs *majorants* (respectivement *minorants*). Ainsi, nous appellerons *infimum* (respectivement *supremum*) les éléments dont la description correspond à la définition suivante :

Définition 6 (infimum - supremum). *Soient $(E,\leq)$ un ensemble ordonné et S une partie non vide de E. L'infimum noté $\bigwedge$ [2] (respectivement supremum noté $\bigvee$ [3]) de S est le plus grand (respectivement le plus petit) élément, s'il existe, de l'ensemble des minorants (resp. des majorants) de S.*

Exemple : Dans la Figure 2.1, l'infimum $\bigwedge$ et le supremum $\bigvee$ représentent le plus grand commun diviseur (PGCD) et le plus petit commun multiple (PPCM), respectivement.

Finalement, nous arrivons à la définition de la structure de treillis construit à partir d'un ensemble d'éléments et d'une relation d'ordre partiel.

Définition 7 (treillis). *Soit $(E,\leq)$ un ensemble ordonné. On dit que $(E,\leq)$ est un treillis si pour tout $x,\ y \in E$: $x \bigwedge y$ et $x \bigvee y$ existent.*

Définition 8 (treillis complet). *Le treillis E est complet si pour toute partie S non vide de E, $\bigvee S$ et $\bigwedge S$ existent.*

Exemple : Le treillis de la Figure 2.1 est un treillis complet.

Propriété 1. *Si E est un treillis fini alors il est complet.*

Définition 9 (demi-treillis). *Un ensemble ordonné $(E,\leq)$ est un sup-demi-treillis[4] (respectivement un inf-demi-treillis[5]) si tout couple d'éléments $x, y \in E$ admet un supremum $x \vee y$ (respectivement un infimum $x \wedge y$).*

---

[2] En anglais "meet"

[3] En anglais "join"

[4] En anglais "join-semi-lattice" ou "sup-semi-lattice".

[5] En anglais "meet-semi-lattice" ou "sub-semi-lattice".

2.1.1.4 Opérateurs de fermeture et correspondance de Galois

Un opérateur de fermeture sur un ensemble ordonné E est une application $\varphi : E \to E$, monotone croissante, extensive et idempotente. Formellement, elle est définie de la manière suivante :

Définition 10 (Opérateur de fermeture). *Soit $(E,\leq)$ un ensemble ordonné. On appelle opérateur de fermeture sur l'ensemble E toute application $\varphi : E \to E$ qui vérifie les trois propriétés suivantes :*

- *idempotence :* $\forall x \in E,\ \varphi(\varphi(x)) = \varphi(x)$,
- *monotonie :* $\forall x, y \in E,\ x \leq y \Rightarrow \varphi(x) \leq \varphi(y)$,
- *extensivité :* $\forall x \in E,\ x \leq \varphi(x)$.

Définition 11 (fermé). *Étant donné un opérateur de fermeture φ sur un ensemble ordonné $(E, \leq)$, un élément $x \in E$ est dit "fermé" si $x = \varphi(x)$. L'élément x est aussi appelé le point fixe de la fermeture.*

Définition 12 (correspondance de Galois). *Soient $(E,\leq_E)$ et $(F,\leq_F)$ deux ensembles ordonnés. La paire de fonctions $f : E \to F$ et $g : F \to E$ définit une correspondance de Galois si la condition suivante est satisfaite :*

$$\forall x \in E, \forall y \in F, f(x) \leq y \Leftrightarrow g(y) \leq x.$$

2.1.1.5 Principe de la dualité

Définition 13 (Relation inverse). *Soit $\leq$ une relation binaire d'un ensemble E vers un ensemble F. On appelle relation inverse de $\leq$, notée $\geq$, la relation binaire de F vers E définie par :* $\forall (x, y) \in E \times F,\ x \leq y \Leftrightarrow y \geq x$.

Si $(E,\leq)$ est partiellement ordonné, la relation inverse $\geq$ est également un ordre partiel sur E. Le principe de la dualité permet de dériver les propriétés duales de $(E,\geq)$ à partir des propriétés de $(E,\leq)$.

Propriété 2. *Soit $\geq$ la relation inverse de $\leq$. Toute assertion sur $\leq$, $\vee$, $\wedge$ reste vraie en remplaçant $\leq$ par $\geq$ et en permutant $\vee$ et $\wedge$.*

2.1.2 Notions fondamentales de l'AFC

L'AFC [30] étudie les structures partiellement ordonnées, connues sous le nom de *treillis de Galois* [5] ou *treillis de concepts*, induites par une relation binaire sur une paire d'ensembles, appelés respectivement individus (*objets*) et traits (*attributs*). L'AFC permet d'identifier des groupements d'objets ayant des attributs en commun, appelés *concepts*, et de les organiser en une hiérarchie conceptuelle (*treillis de concepts*). Elle s'appuie sur les résultats de la théorie des treillis (voir section 2.1.1) dont les fondements mathématiques ont été posés par Birkhoff [8].

2.1.2.1 Contextes Formels

Les données en AFC sont décrites par le biais d'un tableau booléen appelé "*contexte formel*". Formellement, un contexte formel est définit de la manière suivante :

Définition 14 (contexte formel mono-valué). *Un contexte formel est un triplet* $\mathcal{K} = (O, A, I)$ *où* O *est l'ensemble des objets formels,* A *est l'ensemble des attributs formels et* I *est une relation binaire, i.e.,* $I \subseteq O \times A$*, précisant pour chaque objet formel les attributs qui lui sont associés.*

Exemple : La table illustée à droite de la Figure 2.2 présente un contexte mono-valué, appelé "`Humain`", où les objets formels représentent des personnes et les attributs formels représentent certains traits personnels tels que l'âge et l'expérience de travail (e.t.).

2.1.2.2 Construction d'un contexte formel

Le format le plus simple pour décrire un contexte formel est une **table croisée** à deux dimensions où chaque ligne représente un objet formel et chaque colonne représente un attribut formel. L'intersection de la ligne o avec la colonne a contient une croix si et seulement si $(o, a) \in I$. Le type de données qui convient le plus pour

	(a) age≤27	(b) age>27	(c) e.t.≤5	(d) e.t.>5
1	X		X	
2		X	X	
3	X		X	
4	X			X
5		X	X	
6		X		X

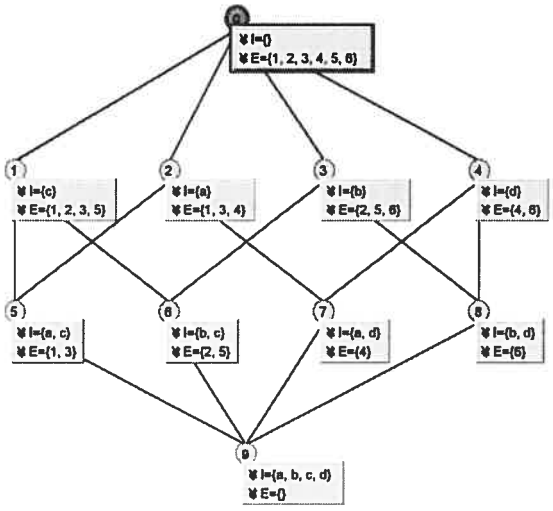


FIG. 2.2 – **Gauche** : Un contexte binaire . **Droite** : Le treillis de concepts associé.

la représentation en mémoire d'un contexte mono-valué est une matrice de bits[6].

Il n'y a pas de restrictions sur la nature des objets et des attributs. On peut considérer des objets physiques, des personnes, des nombres, des processus, des structures, etc. Il est possible de permuter le rôle des objets et des attributs : si $\mathcal{K} = (O, A, I)$ est un contexte, alors (O, A, I^{-1}) est son contexte dual avec $(o, a) \in I \Leftrightarrow (a, o) \in I^{-1}$.

2.1.2.3 Correspondance de Galois dans un contexte formel

Étant donné un contexte formel $\mathcal{K} = (O, A, I)$, on définit les deux fonctions f et g sur ce contexte de la manière suivante :

Définition 15. *La fonction f associe à un ensemble d'objets X, l'ensemble d'attributs Y partagés par tous les objets de X ; tandis que g est la fonction duale pour les ensembles d'attributs :*

[6]Il n'y a pas de type de données standard pour la mémorisation d'un contexte car cela dépend des opérations que l'on veut réaliser qui consistent dans la plupart des cas à calculer les fermetures des ensembles d'objets, respectivement ceux des attributs

- $f : 2^O \rightarrow 2^A,\ f(X) = X' = \{a \in A \mid \forall o \in X, oIa\}$
- $g : 2^A \rightarrow 2^O,\ g(Y) = Y' = \{o \in O \mid \forall a \in Y, oIa\}$

La paire de fonctions (f,g), résumant les liaisons entre les objets et les attributs dans le contexte, définit une correspondance de Galois entre les deux ensembles ordonnés $(2^O,\subseteq)$ et $(2^A,\subseteq)$ (voir la définition 12 de la correspondance de Galois).

Propriété 3. *La paire de fonctions (f,g) de la définition 15 définie une correspondance de Galois entre 2^O et 2^A du contexte formel $\mathcal{K} = (O, A, I)$.*

Exemple : Dans le contexte à gauche de la Figure 2.2, $\{1,3\}' = \{a,c\}$ et $\{b,c\}' = \{2,5\}$.

2.1.2.4 Opérateurs de fermeture dans un contexte formel

Les opérateurs de composition $g \circ f$ et $f \circ g$ sont des opérateurs de fermeture sur les deux ensembles 2^O et 2^A. Par conséquent, chacun des deux opérateurs induit une famille d'ensembles *fermés*, notée $\mathcal{C}^o$ et $\mathcal{C}^a$ respectivement, avec f et g deux applications bijectives entre ces deux familles.

Propriété 4. *Soit un contexte formel $\mathcal{K} = (O, A, I)$. Les compositions des fonctions f et g : $f \circ g : (2^A, \subseteq) \rightarrow (2^A, \subseteq)$ et $g \circ f : (2^O, \subseteq) \rightarrow (2^O, \subseteq)$ sont les opérateurs de fermeture sur 2^A et 2^O, respectivement. Les deux opérateurs de composition sont notés par $''$.*

Ainsi, étant donné le contexte formel $\mathcal{K} = (O, A, I)$ et l'opérateur de fermeture $''$, et selon la définition 10, un sous ensemble d'objets $X \subseteq \mathcal{P}(O)$ (respectivement un sous ensemble d'attributs $Y \subseteq \mathcal{P}(A)$) est fermé si $X'' = X$ (respectivement Y"=Y).

Exemple : Dans le contexte de la Figure 2.2, les ensembles $\{1,3,4\}$ et $\{b,d\}$ sont fermés.

La famille des fermés des objets $\mathcal{C}^o$ et celle d'attributs $\mathcal{C}^a$ munies de la relation d'ordre partiel $\subseteq$ (inclusion ensembliste) constituent deux treillis complets, notés $\mathcal{L}^o$ et $\mathcal{L}^a$. De plus, la fonction $'$ (f ou g) est une bijection entre les deux familles

$\mathcal{C}^o$ et $\mathcal{C}^a$ et définit un isomorphisme entre les treillis respectifs $\mathcal{L}^o$ et $\mathcal{L}^a$. À chaque fermé X de $\mathcal{C}^o$ correspond un fermé unique Y de $\mathcal{C}^a$, tel que $X' = Y$. Dans la section suivante nous allons caractériser ses pairs d'éléments (X, Y) et décrire les différentes structures qui les englobent.

2.1.2.5 Concepts formels, extension et intension

Le couple (X, Y) d'ensembles fermés et mutuellement correspondants où X représente les objets et Y les attributs est appelé *concept (formel).*

Définition 16. *Un concept formel d'un contexte $\mathcal{K}$ est une paire (X, Y) où $X \in 2^O$, $Y \in 2^A$, $X = Y'$ et $Y = X'$. L'ensemble X est appelé extension et Y l'intension du concept formel.*

Exemple : Dans le contexte de la Figure 2.2, le couple $(\{1, 3, 4\}, \{a\})$ est un concept, alors que $(\{2, 5\}, \{b\})$ n'en est pas un.

Selon cette définition, un concept formel a deux dimensions : l'extension X et l'intension Y. Cette description est redondante car chaque dimension détermine l'autre (du fait que $Y = X'$ et $X = Y'$). Toutefois cette définition le rend conforme à la définition traditionnelle de concept en philosophie où «on nomme concept une idée ou représentation de l'esprit qui abrège et résume une multiplicité d'objets empiriques ou mentaux par abstraction et généralisation de traits communs identifiables»[7]. À noter que les deux dimensions suivent la loi de proportionnalité inverse : plus l'extension augmente, plus la compréhension diminue et inversement.

De point de vue schématique, lorsqu'un contexte est décrit par une table binaire, chaque concept formel (X, Y) correspond à une sous table rectangulaire avec un ensemble de lignes X et un ensemble de colonnes Y non nécessairement contigües.

Les concepts formels d'un contexte correspondent aux rectangles maximaux de la table binaire associée. Pour visualiser ces rectangles, il est, parfois, nécessaire de procéder à des permutations de lignes et/ou de colonnes. Ces permutations n'affectent pas le contexte du moment où il n'y a pas d'ordre entre les éléments

[7] http ://fr.wikipedia.org/wiki/Concept#Philosophie

des deux ensembles O et A. Un contexte a en général une multitude de concepts formels. Leur nombre peut être exponentiel en la taille du contexte. L'ensemble de tous les concepts formels d'un contexte $\mathcal{K} = (O, A, I)$ sera noté ici par $\mathcal{C}_\mathcal{K}$. Plusieurs algorithmes ont été proposés pour le calcul de cet ensemble ou encore la relation d'ordre entre les éléments de cet ensemble. Par la suite, nous allons présenter deux approches pour le calcul de l'ensemble $\mathcal{C}_\mathcal{K}$ d'un contexte $\mathcal{K}$ donné.

2.1.2.6 Hiérarchies conceptuelles

La famille $\mathcal{C}_\mathcal{K}$ des concepts formels d'un contexte $\mathcal{K} = (O, A, I)$ est partiellement ordonnée par la relation d'inclusion $\subseteq$ entre intensions/extensions.

Définition 17. *Étant donné (X_1, Y_1) et (X_2, Y_2) deux concepts formels d'un contexte $\mathcal{K} = (O, A, I)$. Le concept (X_1, Y_1) est un* ***sous-concept*** *de (X_2, Y_2) (d'une manière équivalente, (X_2, Y_2) est un* ***super-concept*** *de (X_1, Y_1)) si et seulement si $X_1 \subseteq X_2$ ($Y_2 \subseteq Y_1$). On utilise le signe $\leq$ pour exprimer cette relation. On obtient : $(X_1, Y_1) \leq_\mathcal{K} (X_2, Y_2) \Leftrightarrow X_1 \subseteq X_2 (Y_2 \subseteq Y_1)$.*

Un super-concept immédiat (respectivement le sous-concept immédiat) d'un concept est aussi appelé "*successeur* immédiat (respectivement prédécesseur immédiat).

Définition 18 (successeur immédiat). *Soit $c = (X, Y)$ un concept et $\bar{c} = (\bar{X}, \bar{Y})$ un de ses super-concepts. $\bar{c}$ est dit successeur immédiat de c si et seulement si il n'existe pas un autre concept $\hat{(c)} = (\hat{(X)}, \hat{(Y)})$ tel que $Y \subseteq \hat{(Y)} \subseteq \bar{(Y)}$.*

On définit la face d'un concept par rapport à un successeur immédiat, notée $\delta(c)$, de la manière suivante :

Définition 19 (face d'un concept). *Soit $c = (X, Y)$ un concept et $\bar{c} = (\bar{X}, \bar{Y})$ un successeur immédiat. $\delta(c) = \bar{Y} - Y$.*

La famille des concepts organisée par la relation de succession (précédence) forme un treillis de concepts.

Propriété 5 (treillis de concepts). *L'ensemble de tous les concepts formels de $\mathcal{K} = (O, A, I)$, ordonné par la relation de super-concept (sous-concept) est un treillis complet. Il est noté $\mathcal{L} = \langle \mathcal{C}_\mathcal{K}, \leq_\mathcal{K} \rangle$ et est appelé le treillis de concepts du contexte $\mathcal{K}$.*

Exemple : Le treillis de concepts construit à partir du contexte de la Figure 2.2 est présenté sur la même figure, à droite. À noter que les extensions et les intensions sont montrées sur les nœuds représentant les concepts et identifiables par les ensembles "E" et "I", respectivement.

Le treillis de concepts est muni des opérations dites *infimum* et *supremum* qui sont notées, respectivement, par $\bigwedge$ et $\bigvee$.

Théorème 1. *L'ordre partiel $\mathcal{L} = \langle \mathcal{C}_\mathcal{K}, \leq_\mathcal{K} \rangle$ forme un treillis complet où l'infimum et le supremum sont définis comme suit :*

- $\bigwedge_{i=1}^{k}(X_i, Y_i) = (\bigcap_{i=1}^{k} X_i, (\bigcup_{i=1}^{k} Y_i)'')$,
- $\bigvee_{i=1}^{k}(X_i, Y_i) = ((\bigcup_{i=1}^{k} X_i)'', \bigcap_{i=1}^{k} Y_i)$.

Exemple : Dans le treillis de la Figure 2.2, l'infimum des deux concepts $(\{1,2,3,5\}, \{c\})$ et $(\{1,3,4\}, \{a\})$ est le concept $(\{1,3\}, \{a,c\})$ tandis que le supremum des deux concepts $(\{2,5\}, \{b,c\})$ et $(\{6\}, \{b,d\})$ est le concept $(\{2,5,6\}, \{b\})$.

La couverture supérieure et inférieure d'un concept appartenant au treillis de concepts est définie comme suit :

Définition 20 (couverture inférieure - supérieure). *La couverture supérieure (respectivement inférieure) d'un concept est l'ensemble de tous ses super-concepts (respectivement sous-concepts).*

Une manière de présenter le treillis de concepts qui s'avère très utile en génie logiciel et dans la visulation des treillis de grande taille consiste à supprimer les redondances d'attributs et d'objets dans les intensions et les extensions des concepts, respectivement. En effet, pour un concept $c = (X, Y)$, X est présent dans tous les ancêtres de c et symétriquement, Y apparaît dans tous ses descendants. Ainsi, la suppression des redondances consiste à retirer de l'extension d'un concept tous les

objets formels (respectivement attributs formels) qui apparaîssent dans les extensions (respectivement intensions) des concepts de sa couverture supérieure (respectivement inférieure). La structure obtenue est appelée *treillis d'héritage* ou encore le *treillis des sommets simplifiés* et est notée $\mathcal{L}'$. La Figure 2.3 montre le treillis d'héritage qui correspond au treillis de la Figure 2.2.

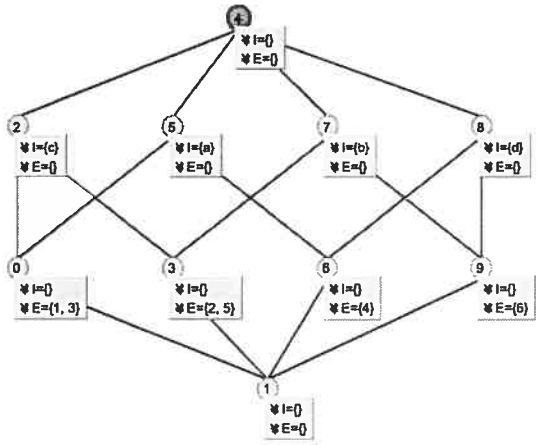


FIG. 2.3 – Le treillis d'héritage correspondant au treillis de la Figure 2.2.

2.1.2.6.1 Concept objet/attribut Dans le treillis de la Figure 2.2, on remarque que chaque objet o est attaché à un unique concept, appelé *concept objet* qui représente le concept minimal (en terme de nombre d'objet) dans le treillis ayant l'objet o dans son extension. De manière similaire, à chaque attribut a correspond un *concept attribut*, noté $\mu(a)$, qui représente le concept maximal dans le treillis ayant l'attribut a dans son intension.

Étant donné un contexte binaire, la paire de fonctions (γ,μ) permet d'obtenir ces concepts particuliers.

Définition 21. *Soit $\mathcal{K} = (O, A, I)$ un contexte binaire et $\mathcal{L} = \langle \mathcal{C}_\mathcal{K}, \leq_\mathcal{K} \rangle$ le treillis complet associé.*

$$\gamma : O \to \mathcal{C}_\mathcal{K},\ \gamma(o) = (\{o\}'', \{o\})$$

Définition 22. *Soit $\mathcal{K} = (O, A, I)$ un contexte binaire et $\mathcal{L} = \langle \mathcal{C}_\mathcal{K}, \leq_\mathcal{K} \rangle$ le treillis complet associé.*

$$\mu : A \to \mathcal{C}_\mathcal{K},\ \mu(a) = (\{a\}', \{a\}'').$$

Exemple : Dans le treillis de la Figure 2.2, $\gamma(\{2\}) = (\{2,5\}, \{b,c\})$, et $\mu(\{d\}) = (\{4,6\}, \{d\})$.

2.1.2.7 Structures dérivées des treillis

De plus que le treillis complet, deux autres structures dérivées ont acquis une large popularité parmi les adeptes de l'AFC, à savoir le *sup-demi-treillis*, appelé aussi "*iceberg*", et les Sous-Hiérarchies de Galois (SHG).

2.1.2.7.1 Le sup-demi-treillis (iceberg) Un iceberg d'un treillis $\mathcal{L}$ est composé des ensembles supérieurs obtenus par les anti-chaînes maximales dans $\mathcal{L}$. Il représente aussi les filtres d'ordre dans $\mathcal{L}=\langle \mathcal{C}_\mathcal{K}, \leq_\mathcal{K} \rangle$.

Définition 23 (filtre d'ordre). *Soient E un ensemble ordonné muni d'un ordre partiel $\leq_E$ et S une partie de E. S est un filtre d'ordre si et seulement si $(x \in S \wedge x \leq y) \to y \in S$. Un idéal d'ordre est défini de façon duale.*

Exemple : Dans le treillis de la Figure 2.2, le fitre d'ordre associé au concept $(\{2,5\}, \{b,c\})$ est composé des concepts $(\{2,5\}, \{b,c\})$, $(\{1,2,3,5\}, \{c\})$, $(\{2,5,6\}, \{b\})$ et $(\{1,2,3,4,5,6\}, \emptyset)$. L'idéal associé au concept $(\{1,3,4\}, \{a\})$ est composé des concepts $(\{1,3,4\}, \{a\})$, $(\{1,3\}, \{a,c\})$, $(\{4\}, \{a,d\})$ et $(\emptyset, \{a,b,c,d\})$.

Intuitivement, l'iceberg est obtenu en coupant le treillis $\mathcal{L}=\langle \mathcal{C}_\mathcal{K}, \leq_\mathcal{K} \rangle$ de façon horizontale en deux parts. Les restrictions sur la cardinalité de l'extension des concepts est le critère de coupe le plus populaire. Ce critère est appelé *fréquence/support* d'un concept. Étant donné un concept $c = (X, Y)$, la fonction ν retourne la fréquence correspondante de la manière suivante :

Définition 24 (application ν). *Soient $\mathcal{K} = (O, A, I)$ un contexte formel et $c = (X, Y)$ un concept du treillis $\mathcal{L}=\langle \mathcal{C}_\mathcal{K}, \leq_\mathcal{K} \rangle$ correspondant.*

$$\nu : \mathcal{C}_\mathcal{K} \to \mathbb{R},\ \nu(c) = \|X\|/\|O\|.$$

Ainsi, pour obtenir un iceberg à partir d'un treillis $\mathcal{L}=\langle \mathcal{C}_\mathcal{K}, \leq_\mathcal{K} \rangle$, il faut se donner un seuil de fréquence $\alpha \in]0,1]$ et filtrer les concepts pour ne garder que ceux vérifiant la condition de fréquence $\nu(c) \geq \alpha$. De cette manière, le treillis est coupé en deux parts. La partie supérieure, notée $\bar{\mathcal{L}}^\alpha=\langle \bar{\mathcal{C}}^\alpha, \leq_\mathcal{K} \rangle$, où $\bar{\mathcal{C}}^\alpha=\{c | c \in \mathcal{C}_\mathcal{K}, \nu(c) \geq \alpha\}$ est appelée l'iceberg [80] ou encore le $\alpha-$iceberg.

Définition 25 (α-iceberg). *Le α-iceberg, $\alpha \in]0,1]$, d'un treillis de Galois et le sup-demi-treillis $\bar{\mathcal{L}}^\alpha = \langle \bar{\mathcal{C}}^\alpha, \leq_\mathcal{K} \rangle$, où $\bar{\mathcal{C}}^\alpha$ est l'ensemble des concepts α-fréquent, c'est-à-dire, $\bar{\mathcal{C}}^\alpha = \{c | c \in \mathcal{C}, \nu(c) \geq \alpha\}$.*

Exemple : L'iceberg de la Figure 2.4 est obtenu à partir du treillis de la Figure 2.2 avec un seuil de fréquence $\alpha = 0.20$.

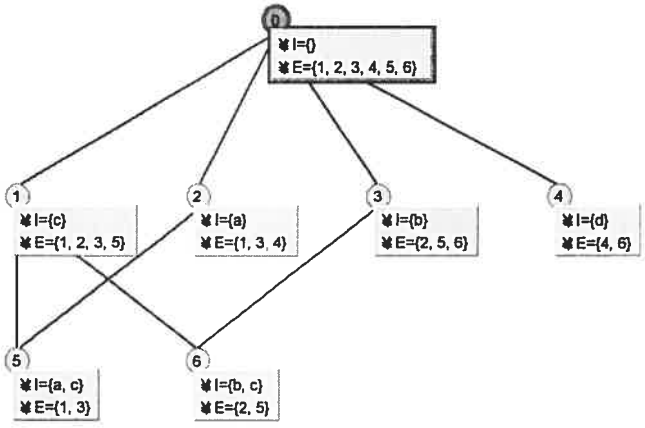


FIG. 2.4 – L'iceberg obtenu à partir du treillis de la Figure 2.2 avec un seuil de fréquence 20%.

L'iceberg est généralement la structure la plus intéressante du fait qu'elle inclut les concepts les plus généraux et exclut ceux qui sont plus spécifiques. Ces concepts généraux sont très utiles dans plusieurs applications de l'AFC telles que le génie logiciel où l'on s'interesse aux groupes d'objets/attributs les plus représentatifs ou en fouille de données où il s'agit d'extraire des patrons fréquents de relations (implications) dans les données brutes (voir Chapitre 2, Section 2.3.1.1).

2.1.2.7.2 La sous-Hiérarchie de Galois Les travaux effectués dans le cadre des applications de l'AFC en génie logiciel, notamment la restructuration des hiérarchies de classes des systèmes orientés objet [32, 19] ont montré l'importance d'une autre sous-structure d'un treillis complet appelée *sous-hiérarchie de Galois* (SHG) [18, 33]. Cette sous structure constitue une restriction du treillis aux concepts objets et concepts attributs, c-à-d, les concepts minimaux (maximaux) parmi ceux dont l'extension (intension) inclut un objet donné (attribut).

Définition 26. *Étant donné un treillis $\mathcal{L}$. La SHG correspondante est composée des concepts $\mathcal{L}' \setminus \{c \in \mathcal{L} \mid c = (\emptyset, \emptyset)\}$ où $\mathcal{L}'$ est le treillis d'héritage associé à $\mathcal{L}$. L'ordre entre les concepts restants est préservé.*

Exemple : La Figure 2.5 montre la SHG qui correspond au contexte de la Figure 2.2.

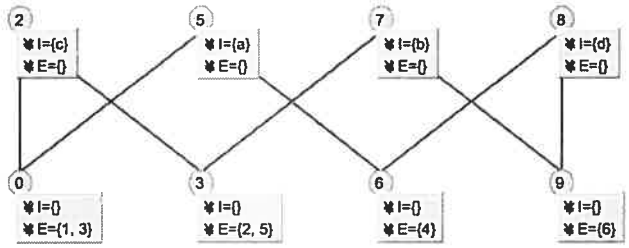


FIG. 2.5 – La SHG obtenue à partir du contexte de la Figure 2.2.

La complexité de la SHG est inférieure au treillis complet. On obtient ainsi une hiérarchie plus compacte qui préserve les caractéristiques de factorisation maximale. L'inconvénient est que la structure n'est plus nécessairement un treillis et que certaines abstractions potentiellement utiles sont éliminées.

Après avoir présenté quelques notions fondamentales de l'AFC, notamment le treillis de concepts construit à partir d'une relation binaire nous allons maintenant montré d'autres types de relations et la manière avec laquelle l'AFC procède pour former des concepts.

2.1.3 Données non binaires

2.1.3.1 Ensemble étendu d'attributs

Depuis l'adoption des treillis de concepts comme un outils d'analyse de données, leur cadre théorique a fait l'objet de nombreuses extensions dont l'intérêt consiste à remplacer les descriptions binaires des individus (conjonction d'attributs binaires) par d'autres formes de descriptions plus expressives. Le but est de pouvoir traiter des données concrêtes telles que celles de type objet-attribut-valeurs issues des bases de données et de la modélisation des problèmes concrets.

Ainsi, beaucoup de travaux [20, 31, 46] ont émergé explorant les possibilités de traduire les différents types de données en variables binaires, notamment ceux basés sur les *échelles conceptuels* [30]. D'autres travaux ont reconsidéré la définition même de certaines constructions fondamentales de la théorie des treillis, telles que la *correspondance de Galois*, dans le cas où les objets admettent des structures complexes, éventuellement floues [31], probabilistes [20] ou ensembles bruts [46]. En AFC, les attributs non binaires (e.g., numériques, ordinaux, catégoriques, etc.) sont formulés à travers des contextes pluri-valués [30].

Définition 27 (contexte pluri-valué). *Un contexte pluri-valué est un quadruplet* $\mathcal{K} = (O, A, V, I)$ *où* O *et* A *sont des ensembles d'objets et d'attributs respectivement,* V *est un ensemble de valeurs, et* $I \subseteq O \times A \times V$ *est une relation ternaire ayant des tuples sous la forme* $(o\ a\ v)$ *qui signifie "l'objet* o *a la valeur* v *pour l'attribut* a*".*

La Table 2.1 montre un contexte pluri-valué, appelé *Humain*. Il est composé de six objets (1, 2, ..., 6) représentant des individus et deux propriétés, *âge* et *travail* exprimant, respectivement, l'âge et le nombre d'années d'expérience de chaque individu.

Dans le but de traiter un contexte pluri-valué $\mathcal{K} = (O, A, V, I)$, celui-ci est d'abord transformé en un contexte mono-valué (binaire) équivalent $\mathcal{K}^d$ appelé aussi le contexte *dérivé*. Pour ce faire, l'AFC possède une technique, appelée "*scaling*", que nous allons décrire dans la section suivante.

	o_1	o_2	o_3	o_4	o_5	o_6
âge	25	28	22	26	33	35
travail	4	2	1	8	4	10

TAB. 2.1 – Contexte pluri-valué *Humain.*

2.1.3.2 Scaling conceptuel

D'une manière générale, il n'y a pas un moyen "automatique" pour dériver des structures conceptuelles à partir de contextes pluri-valués. La première approche qui consiste à dériver des treillis à partir de ce genre de contextes a été proposée par Ganter *et al.* dans [29]. Dès lors, une grande variété d'applications se sont appuyées sur cette méthode notamment celles qui traitent des données brutes en provenance des bases de données [62].

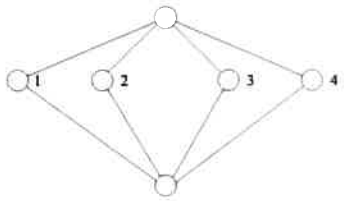
Le scaling conceptuel [30] fournit un cadre théorique complet pour la transformation de n'importe quel contexte pluri-valué en un contexte binaire équivalent. En effet, le processus de scaling associe à chaque attribut pluri-valué un context binaire appelé *échelle conceptuelle.*

2.1.3.2.1 Échelle

Définition 28 (Échelle). *Une* ***échelle*** *pour un attribut a d'un contexte pluri-valué donné est un contexte mono-valué $\mathcal{K}_a = (V_a, P_a, I_a)$ où les objets formels V_a sont les valeurs de l'attribut a tandis que les attributs formels P_a sont des propriétés distinguées de ces valeurs. Les objets de l'échelle sont appelés* ***valeurs de l'échelle*** *alors que les attributs sont appelés* ***attributs de l'échelle.***

Quelques contextes élémentaires sont souvent utilisés comme des échelles. Nous citons à titre illustratif l'échelle nominale, ordinale, biordinale et dichotomique [30]. Par exemple, l'échelles nominale, notée $\mathcal{N}_n = (n, n, =)$, est utilisée avec les attributs pluri-valués dont les valeurs s'excluent mutuellement. Un attribut pluri-valué pouvant avoir les valeurs masculin, féminin ou neutre doit subir un scaling nominal. Ainsi les extensions des concepts définissent un **partitionnement** de l'ensemble d'objets de l'échelle tel que illustré à droite de la Figure 2.2.

	1	2	3	4
1	X			
2		X		
3			X	
4				X



TAB. 2.2 – **Gauche :** Échelle nominale $\mathcal{N}_4$. **Droite :** Le treillis associé (Adapté de [30]).

2.1.3.2.2 Contextes dérivés par des échelles conceptuelles à chaque échelle correspondant à un attribut a est associé un contexte $\mathcal{K}_a = (V_a, P_a, J_a)$ où les valeurs v_i de l'attribut a sont les objets formels tandis que les propriétés essentielles de ces valeurs deviennent les attributs formels a_j^d. À titre d'exemple, dans les échelles ordinales, les propriétés sont des prédicats permettant de comparer les valeurs à une constante dans un domaine (exemple, "age $\geq$ 20").

Les échelles correspondent aux différentes dimensions de la description de l'objet formel. Essentiellement, une échelle décrit d'une manière hiérarchique des abstractions utiles d'une dimension donnée, appelées *concepts de l'échelle*. Ces abstractions correspondent à des sous-ensembles de valeurs du co-domaine de l'attribut a (par exemple,"âge $\geq$ 20 et $\leq$ 27"), notés $cod(a) = V_a$. Par la suite, nous allons noter l'extension de l'attribut de l'échelle a_j^d, en terme de valeurs, par $a_j^{d'}$.

2.1.3.2.3 Treillis de l'échelle Le treillis de l'échelle $\mathcal{L}_a$ qui correspond au contexte $\mathcal{K}_a$ permet un regroupement des valeurs de l'attributs en sous-ensembles significatifs pour l'utilisateur. Les concepts échelle dans $\mathcal{L}_a$ représentent les sous-ensembles de valeurs de a. Dans le cas où la cardinalité de l'ensemble $cod(a)$ est élevée (cas des valeurs d'une colonne dans une base de données), le treillis $\mathcal{L}_a$ peut être difficile à obtenir d'une manière directe, c'est à dire, à partir de $\mathcal{K}_a$. Toutefois, la relation d'inclusion entre les extensions des différents attributs de l'échelle peut être utilisée pour restaurer la structure de $\mathcal{L}_a$. Dans ce cas particulier, le contexte formel utilisé est $\mathcal{K}_a^f = (P_a, P_a, \leq_f)$, où $a_j^d \leq_f a_l^d$ si et seulement si $a_j^{d'} \subseteq_f a_l^{d'}$.

Exemple : La Figure 2.6 présente des treillis échelles pour les attributs pluri-

valués *âge* et *experience de travail* correspondant au contexte *Humain*. Les seuils 27 et 5 ont été arbitrairement choisis pour les attributs *âge* et *travail* respectivement.

FIG. 2.6 – Échelles pour *âge* et *expérience de travail* suivant la Table 2.1.

2.1.3.2.4 Scaling Le but de tout processus de sclaing en AFC est l'obtention, à partir d'un contexte pluri-valué $\mathcal{K} = (O, A, V, I)$, le contexte formel dérivé $\mathcal{K}^d$ ayant les mêmes objets que le contexte de départ et des attributs en provenance de différentes échelles représentant des propriétés "significatives". Le mot "significatives" fait référence à l'interprétation des données qui ne peut être établie que par un expert du domaine. Cette interprétation est toujours faite dans un but et fondée sur des considérations théoriques [62].

Le principe du scaling conceptuel consiste à affecter à chaque attribut pluri-valué $a \in A$ une échelle conceptuelle $\mathcal{K}_a$ qui est un contexte formel mono-valué $\mathcal{K}_a = (V_a, P_a, I_a)$. Ainsi, chaque objet verra sa valeur de l'attribut non binaire a remplacée par un ensemble de valeurs binaires, a_i^d. Un objet o de $\mathcal{K}$ obtient a_i^d chaque fois que a_i^d inclut la valeur de a dans o. Le résultat est un contexte dérivé $\mathcal{K}^d$ qui peut être considéré comme étant une concaténation (une apposition comme dans [30]) d'un ensemble de petits contextes $\mathcal{K}_a^d$, un par attribut pluri-valué. À partir du contexte dérivé, on peut construire le treillis de concepts de manière classique comme en [96]. Le treillis du contexte dérivé est considéré comme étant la structure conceptuelle du contexte pluri-valué.

Exemple : Le contexte à gauche de la Figure 2.2, appelé $Humain^d$, a été obtenu par le scaling du contexte *Humain* de la Table 2.1 en utilisant les échelles conceptuelles présentées dans la Figure 2.6.

2.1.3.2.5 Construction du treillis par scaling Le treillis $\mathcal{L}^d$ associé au contexte dérivé $\mathcal{K}^d$ peut être obtenu directement à partir de ce contexte en utilisant n'importe quelle procédure usuelle de construction de treillis. Il est évident que la forme exacte du treillis $\mathcal{L}^d$ dépend des échelles choisies pour les différents attributs muti-valués. Par exemple, le treillis du contexte dérivé $Human^d$ est donné à droite de la Figure 2.2.

Un autre intérêt des échelles conceptuelles réside dans la possibilité de les utiliser pour la visualisations des treillis, en particuliers ceux de grande taille (comme dans le système TOSCANA [94]). De plus, le treillis $\mathcal{L}^d$ peut être obtenu directement à partir des treillis $\mathcal{L}_a^d$ associés aux différents attributs pluri-valués du contexte initial $\mathcal{K}$ en utilisant une version légèrement modifiée de l'algorithme ASSEMBLER décrit dans [91]. Par exemple, dans le cas du contexte dérivé $Humain^d$, les correspondances entre les treillis échelles de ses attributs pluri-valués *age* et *travail* (voir Figure 2.6) et le treillis $\mathcal{L}^d$ qui lui correspond (voir Figure 2.2, à droite) sont faciles à établir.

2.2 Construction du treillis de Galois

La construction du treillis à partir du contexte a toujours été un défi calculatoire vu que le nombre de concepts dans le treillis peut être exponentiel en fonction du nombre d'attributs du contexte. Toutefois, en pratique, seulement un petit nombre de concepts sont produits d'où l'intérêt de chercher des méthodes permettant de découvrir d'une manière efficace les concepts et, si nécessaire, de les organiser en une hiérarchie.

La construction d'un treillis est composée de deux principales tâches qui peuvent être exécutées de manière simultanée ou séquentielle, à savoir, la découverte des concepts et le calcul de la relation de couverture. Dans la littérature de l'AFC, il y a une grande variété d'algorithmes[8] dédiés à ces deux tâches séparément ou conjointement. Toutefois, une distinction majeure entre ces algorithmes réside dans

[8]Voir [47] pour une étude comparative.

la manière d'acquérir les données d'entrée. Suivant la dichotomie actuelle, trois paradigmes d'algorithmes sont considérées :

- **Algorithmes batch** [10, 58, 28] : représente la première génération des algorithmes de construction de treillis. Ils considèrent que les données (contexte formel) sont connues à l'avance. L'évolution des données (ajout objet/attribut au contexte) entraîne la reconstruction du treillis de nouveau.
- **Algorithmes incrémentaux** [36, 83] : se sont apparus pour remédier au problème de la reconstruction du treillis dans le cadre des contextes dynamiques. En effet, suite à une modification du contexte, ces algorithmes effectuent des mises à jour locales du treillis correspondant. À noter qu'un algorithme incrémental peut simuler un algorithme batch en réalisant une suite d'ajouts d'objets/attributs à un contexte initialement vide.
- **Algorithmes d'assemblage** [91] : Les travaux [36, 83] sur les performances des algorithmes incrémentaux ont abouti à une troisième catégorie d'algorithmes qui généralise l'incrémentation à des ensembles d'objets/attributs. Ces algorithmes se basent sur l'apposition/sous-position de contextes et les demi-produits de treillis, pour définir une opération binaire sur les treillis complets, appelée ASSEMBLAGE permettant de construire un treillis à partir des deux treillis associés aux deux parties du contexte. Cette technique peut être utilisée dans le cadre de la stratégie "*diviser pour régner*" de construction de treillis.

À noter que les sous structures de treillis, mentionnées plus haut, ont aussi bénéficie de développement algorithmique. Ainsi, des techniques particulières ont été élaborées pour ce type de structures permettant de traiter des problèmes spécifiques à un faible coût. Concernant les SHG par exemple, un ensemble d'algorithmes efficaces, batch et incrémentaux, ont été proposés dans la littérature tels que ceux de Godin [32]. Les icebergs quant à eux restent proches des treillis complet et par conséquent peu d'algorithmes ont été, explicitement, conçus pour eux. Parmi les méthodes dédiées aux icebergs citons TITANIC [80] et la paire d'algorithmes appelée MAGALICE (MAGALICE-O [68] et MAGALICE-A [56] qui seront

détaillés dans le chapitre 5.

Dans le cadre de cette thèse, nous avons utilisé le paradigme incrémental. Les algorithmes incrémentaux considèrent le tableau de la relation binaire une ligne à la fois. Cette approche est fort intéressante parce qu'on n'a pas besoin de recalculer de nouveau les éléments du treillis de Galois chaque fois qu'un nouvel objet est considéré. Par conséquent, les algorithmes incrémentaux permettent beaucoup plus le maintient de l'intégrité du treillis suite à l'ajout d'un nouvel objet/attribut au contexte que la construction proprement dite du treillis.

Godin *et al.* [36] a proposé une procédure incrémentale qui permet de modifier de manière locale la structure d'un treillis (insérer de nouveaux concepts, compléter des concepts existants, détruire des liens redondants, etc.) tandis qu'une grande partie du treillis reste inchangée. Un autre algorithme incrémental a été conçu par Carpineto et Romano [12]. De notre part, nous avons proposé un cadre théorique décrivant toutes les restructurations requises sur le treillis lors de l'ajout d'un objet/attribut au contexte correspondant [86]. À partir de ce cadre, nous avons mis au point deux méthodes, à savoir MAGALICE-O [68] et MAGALICE-A [56] permettant la mise à jour incrémentale d'un iceberg par ajout d'objet et ajout d'attribut, respectivement.

2.2.1 Incrémentalité par objet

Une manière de construire le treillis $\mathcal{L}$ consiste à progressivement ajouter un objet o_i au treillis $\mathcal{L}_{i-1}$ (qui correspond au contexte $\mathcal{K} = (\{o_1, ..., o_{i-1}\}, A, I)$), en commençant par le seul objet o_1 et en faisant à chaque étape (ajout) une série de mises à jour structurelles. Dans ce qui suit, nous allons considérer l'exemple de l'ajout du nouvel objet $(\{9\}, \{cdfgh\})$ au contexte formel de la Table 2.3 dont le treillis est illustré par la Figure 2.7.

	a	b	c	d	e	f	g	h
1	0	1	1	1	0	0	1	1
2	1		1	0	0	0	0	0
3	0	0	0	1	1	1	1	1
4	0	0	0	0	0	0	1	0
5	0	0	0	0	1	1	0	1
6	1	1	1	1	0	0	0	0
7	0	1	1	1	0	0	0	0
8	0	0	0	1	0	0	0	0

TAB. 2.3 – Un contexte formel.

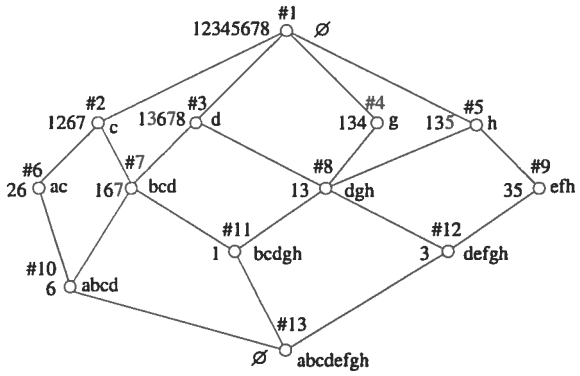


FIG. 2.7 – Le treillis de concepts du contexte de la Table 2.3.

2.2.1.1 Fondements théoriques :

Nous allons rappeler les aspects théoriques importants de l'approche incrémentale définit dans [90] et [86] nécessaires à la mise au point des algorithmes incrémentaux qui seront présentés dans ce chapitre et au chapitre 5. Pour plus de détails, le lecteur est invité à consulter les deux sources principales sus-citées.

Soient le contexte $\mathcal{K} = (O, A, I)$ et le contexte $\mathcal{K}^+ = (O^+, A, I^+)$ obtenus par l'insertion du nouvel objet o dans $\mathcal{K}$ tel que $O^+ = O \cup \{o\}$ et $I^+ = I \cup (\{o\} \times o')$. Le treillis $\mathcal{L}$=$\langle \mathcal{C}, \leq \rangle$ (respectivement $\mathcal{L}^+$=$\langle \mathcal{C}^+, \leq^+ \rangle$) dénote le treillis qui correspond au contexte $\mathcal{K}$ (respectivement $\mathcal{K}^+$).

L'approche incrémentale s'appuie sur le fait que si $c = (X, Y)$ est un concept de $\mathcal{L}$, alors $Y \cap \{o\}'$ est fermé et correspond à un concept de $\mathcal{L}^+$.

Propriété 6. *: L'ensemble $\mathcal{C}^a$ est fermé par l'intersection.*

La mise à jour d'un treillis $\mathcal{L}$ suite à l'ajout du nouvel objet o au contexte associé consiste à calculer l'intersection des intensions des concepts de $\mathcal{L}$ avec $\{o\}'$. De ce fait, deux types d'intersections surgissent : celles qui existent déjà dans $\mathcal{L}$ et celles qui sont nouvelles. Ainsi, les concepts de $\mathcal{L}$ sont répartis en trois catégories : les concepts dits "*modifiés*", dénotés $\mathbf{M}(o)$, qui sont le résultat des intersections existantes, les concepts dits "*inchangés*", dénotés $\mathbf{U}(o)$, qui demeurent inchangés, et les concepts dits "*géniteurs*", dénotés $\mathbf{G}(o)$, qui donnent naissance à de nouveaux concepts, dénotés $\mathbf{N}^+(o)$ (nouvelles intersections). Dans $\mathcal{L}^+$ ces mêmes ensembles sont dénotés $\mathbf{M}^+(o)$, $\mathbf{U}^+(o)$ et $\mathbf{G}^+(o)$, respectivement.

Exemple : La Figure 2.8 montre le treillis $\mathcal{L}^+$ obtenu suite à l'ajout de l'objet $(\{9\}, \{cdfgh\})$[9]. Les modifiés, géniteurs et inchangés dans $\mathcal{L}$ et $\mathcal{L}^+$ sont :

- $\mathbf{G}(9) = \{c_{\#7}, c_{\#9}, c_{\#11}, c_{\#12}, c_{\#13}\}$,
- $\mathbf{M}(9) = \{c_{\#2}, c_{\#3}, c_{\#4}, c_{\#5}, c_{\#8}\}$,
- $\mathbf{U}(9) = \{c_{\#10}, c_{\#6}\}$,
- $\mathbf{G}^+(9) = \{c_{\#7}, c_{\#9}, c_{\#11}, c_{\#12}, c_{\#13}\}$,
- $\mathbf{M}^+(9) = \{c_{\#2}, c_{\#3}, c_{\#4}, c_{\#5}, c_{\#8}\}$,

[9]Par mesure de lisibilité, nous donnerons les ensembles d'objets/attributs sans séparateurs.

- $\mathbf{N}^+(9) = \{c_{\#14}, c_{\#15}, c_{\#16}, c_{\#17}, c_{\#18}\}$,
- $\mathbf{U}^+(9) = \mathbf{U}(9)$.

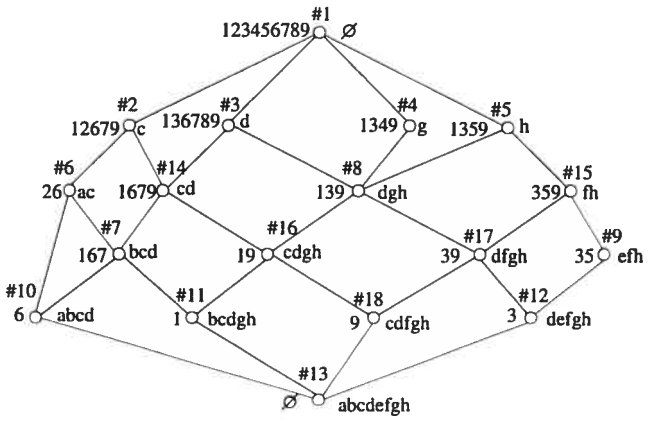


FIG. 2.8 – Le treillis $\mathcal{L}^+$ obtenu après restructuration du treillis $\mathcal{L}$ de la figure 2.7 suite à l'ajout du nouvel objet $(\{9\}, \{cdfgh\})$.

La tâche de restructuration du treillis $\mathcal{L}$ consiste à déterminer les ensembles de géniteurs $\mathbf{G}(o)$ afin de pouvoir créer les nouveaux concepts et les modifiés $\mathbf{M}(o)$ afin de mettre à jour les extensions correspondantes. Il faut également calculer la couverture des nouveaux concepts.

Pour ce faire, nous allons d'abord définir les fonctions sur lesquelles repose la définition de ces ensembles ensuite étudier la relation de couverture dans $\mathcal{L}^+$.

2.2.1.2 Définitions

Les deux application σ et γ établissent des liaisons entre les deux treillis $\mathcal{L}$ et $\mathcal{L}^+$ comme suit :

Définition 29 (application σ et γ).

- *Pour un concept c du treillis $\mathcal{L}$, la fonction σ retourne le concept c^+ de $\mathcal{L}^+$ ayant une intension identique à celle de c.*

$$\sigma : \mathcal{C} \rightarrow \mathcal{C}^+, \sigma(X, Y) = (Y', Y).$$

- *Soit un concept c^+ de $\mathcal{L}^+$. La fonction γ retourne le concept c de $\mathcal{L}$ ayant une extension égale à c^+ modulo o.*

$$\gamma : \mathcal{C}^+ \rightarrow \mathcal{C}, \gamma(X,Y) = (X \setminus \{o\}, (X \setminus \{o\})').$$

Exemple : Considérons le treillis $\mathcal{L}$ donné par la Figure 2.7 et le treillis augmenté $\mathcal{L}^+$ donné par la Figure 2.8. $\sigma(c_{\#8}) = c_{\#8}$ et $\gamma(c_{\#17}) = c_{\#12}$.

Formellement, les catégories de concepts produites suite à l'ajout d'un nouvel objet sont définies comme suit :

Les nouveaux concepts dans $\mathcal{L}^+$ sont les concepts qui une fois le nouvel objet o retiré de leurs extensions, on obtient des extentions existantes dans $\mathcal{L}$ (leurs géniteurs).

Définition 30 ($\mathbf{N}^+(o)$). *L'ensemble des nouveaux concepts dans $\mathcal{L}^+$ est :*

$$\mathbf{N}^+(o) = \{c = (X,Y) | c \in \mathcal{C}^+, o \in X; (X \setminus \{o\})'' = X \setminus \{o\}\}$$

Exemple : Soient le treillis de la Figure 2.8 et le nouveau concept $c^+_{\#15} \in \mathbf{N}^+(9)$. $c^+_{\#15} = (395, fh)$, $X \setminus 9 = 35$, $(X \setminus \{9\})'' = 35$ qui correspond au concept géniteur $c_{\#9}$

Les concepts modifiés conservent leurs intensions lors du passage de $\mathcal{L}$ vers $\mathcal{L}^+$ tandis que leurs extensions sont mises à jour par l'ajout du nouvel objet o.

Définition 31 ($\mathbf{M}^+(o)$ et $\mathbf{M}(o)$). *Les ensembles de concepts modifiés dans $\mathcal{L}^+$ et $\mathcal{L}$ sont :*

- $\mathbf{M}^+(o) = \{c = (X,Y) | c \in \mathcal{C}^+, o \in X; (X \setminus \{o\})' = Y\}$
- $\mathbf{M}(o) = \{c = (X,Y) | c \in \mathcal{C}, \exists\ \bar{c} \in \mathbf{M}^+(o); c = \gamma(\bar{c})\}$

Exemple : Soit le treillis de la Figure 2.8. Le concept $c^+_{\#3} \in \mathbf{M}^+(9)$. $c^+_{\#3} = (136789, d)$, $X \setminus 9 = 13678$, $(X \setminus 9 = 13678)' = d$ qui représente l'intension du concept $c_{\#3}$ avec $c_{\#3} \in \mathbf{M}(9)$ et $\gamma(c_{\#3}) = c_{\#3}$.

Définition 32 ($\mathbf{G}^+(o)$ et $\mathbf{G}(o)$). *Les ensembles de concepts géniteurs dans $\mathcal{L}^+$ et $\mathcal{L}$ sont :*

- $\mathbf{G}^+(o) = \{c = (X,Y) | c \in \mathcal{C}^+, o \notin X; (X \cup \{o\})'' = X \cup \{o\}\}$
- $\mathbf{G}(o) = \{c = (X,Y) | c \in \mathcal{C}, Y \not\subseteq \{o\}'; (X \cup \{o\})'' = X \cup \{o\}\}$ *avec* $''$ *est calculé dans* $\mathcal{K}^+$

Exemple : Soit le treillis de la Figure 2.8. Le concept $c^+_{\#9} \in \mathbf{G}^+(9)$ car $c^+_{\#9} = (35, efh)$, $X \cup \{9\} = 359$ et $(X \cup \{9\})'' = (359)'' = 359$. De même, le concept $c_{\#11} \in \mathbf{G}(9)$ car $c_{\#11} = (1, bcdgh)$ et $bcdgh \not\subseteq \{9\}'$, $X \cup \{9\} = 19$, $(X \cup \{9\})'' = (19)'' = (cdgh)' = 19$.

Le concept géniteur peut être aussi définit comme étant le concept de $\mathcal{L}$ dont l'intension n'est pas incluse dans $\{o\}'$ mais qui est égale à la fermeture de l'intersection avec $\{o\}'$. C'est ce que exprime la propriété suivante :

Propriété 7 ($\mathbf{G}(o)$). *L'ensemble des concepts géniteurs dans le treillis* $\mathcal{L}$ *est :*

$$\mathbf{G}(o) = \{c = (X,Y) | c \in \mathcal{C}, Y \not\subseteq \{o\}'; Y = (Y \cap \{o\})''\}$$

Finalement, les concepts inchangés sont définit comme suit :

Définition 33 ($\mathbf{U}^+(o)$ et $\mathbf{U}(o)$). *Les ensembles des concepts inchangés dans* $\mathcal{L}^+$ *et* $\mathcal{L}$ *sont :*

- $\mathbf{U}^+(o) = \{c = (X,Y) | c \in \mathcal{C}^+; \gamma(c) = c\}$
- $\mathbf{U}(o) = \{c = (X,Y) | c \in \mathcal{C}; \sigma(c) = c\}$

Exemple : Soient le treillis initial de la Figure 2.8 et le treillis obtenu après insertion du nouvel objet ({9},{cdfgh}). Le concept $c^+_{\#10} \in \mathbf{U}^+(9)$. En effet, $\gamma(c^+_{\#10}) = c_{\#10}$. Le concept $c_{\#6} \in \mathbf{U}(9)$ car $\sigma(c_{\#6}) = c^+_{\#6}$.

Comme le calcul de $\mathcal{L}^+$ est obtenu par le calcul de l'intersection de chaque concept c de $\mathcal{L}$ avec le concept objet $\nu(o)$, on définit l'application $\mathcal{Q}$ qui lie les éléments de $\mathcal{C}$ à l'ensemble des sous-ensembles de A de la manière suivante :

Définition 34. $\mathcal{Q} : \mathcal{C} \rightarrow 2^A, \mathcal{Q}(X,Y) = Y \cap \{o\}'$

La fonction $\mathcal{Q}$ induit une relation d'équivalence sur $\mathcal{C}$. La classe d'équivalence d'un concept c, notée $[c]_\mathcal{Q}$, est donnée par :

$$[c]_\mathcal{Q} = \{\bar{c} \in \mathcal{C} \mid \mathcal{Q}(c) = \mathcal{Q}(\bar{c})\}$$

Exemple : Lors de l'ajout de l'objet $(\{9\}, \{cdfgh\})$, $\mathcal{Q}_1(c_{\#6}) = \{c\}$. $[c_{\#7}] = \{c_{\#7}, c_{\#10}\}$.

De plus, l'ensemble des classes d'équivalence $\mathcal{C}_{/\mathcal{Q}}$ est ordonné par la relation $\leq_{/\mathcal{Q}}$:

Définition 35. *(relation d'équivalence $\leq_{/\mathcal{Q}}$) Soient c et $\bar{c}$ deux concepts de $\mathcal{C}$.*

$$[c]_{\mathcal{Q}} \leq_{/\mathcal{Q}} [\bar{c}]_{\mathcal{Q}} \Leftrightarrow \mathcal{Q}(\bar{c}) \subseteq \mathcal{Q}(c)$$

La structure partiellement ordonnée résultante $\mathcal{L}_{/\mathcal{Q}}$ est un treillis complet.

Propriété 8. *$\mathcal{L}_{/\mathcal{Q}} = \langle \mathcal{C}_{/\mathcal{Q}}, \leq_{/\mathcal{Q}} \rangle$ est un treillis complet.*

Chaque classe d'équivalence dans $\mathcal{C}_{/\mathcal{Q}}$ possède un élément maximal $\bar{c}$ unique dont l'intension est égale à la fermeture de la valeur $\mathcal{Q}(c)$ de n'importe quel concept c de la classe d'équivalence.

Propriété 9. *$\forall c \in \mathcal{C}, \exists\ \bar{c} = max([c]_{\mathcal{Q}})$, où $\bar{c} = (\bar{X}, \bar{Y})$ avec $\bar{Y} = (\mathcal{Q}(c))''$. l'opérateur $''$ est calculé dans $\mathcal{K}$.*

Exemple : Soit le treillis de la Figure 2.8. Le concept $c_{\#7} = (167, bcd)$, $Y = bcd$, {9}'=cdfgh, $Y \cap \{9\}' = cd$, $(Y \cap \{9\}')'' = (167)' = bcd$. Par conséquent, $max([c_{\#7}]_{\mathcal{Q}}) = c_{\#7}$.

L'ensemble des maximas des classes d'équivalence est noté $\mathbf{E}(o)$. À partir de la propriété 7, et du fait que $\mathbf{M}(o) \subseteq \mathbf{E}(o)$, on déduit que les concepts maxima des classes sont les concepts géniteurs et modifiés.

Propriété 10 (ensemble des maxima). *L'ensemble des concepts maxima des classes d'équivalence dans le treillis $\mathcal{L}$ sont $\mathbf{E}(o) = \mathbf{G}(o) \cup \mathbf{M}(o)$*

Étant donné que nous avons caractérisé toutes les catégories de concepts ainsi que la manière de les obtenir, il nous reste à déterminer le moyen d'établir les liens de couverture qui doivent être créés dans $\mathcal{L}^+$, c'est à dire, la relation de précédence $\prec^+$.

On définit d'abord l'application χ et χ^+ qui envoie un concept c de $\mathcal{L}$ vers un concept de $\mathcal{L}$ dont l'intension est égale à la fermeture de $\mathcal{Q}(c)$ et un concept de $\mathcal{L}^+$ dont l'intension est égale à $\mathcal{Q}(c)$, respectivement.

Définition 36 (χ^+ et χ). *Les fonctions χ^+ et χ sont définit par :*

- $\chi^+ : \mathcal{C} \rightarrow \mathcal{C}^+, \chi^+(c) = (\mathcal{Q}(c)', \mathcal{Q}(c))$
- $\chi : \mathcal{C} \rightarrow \mathcal{C}, \chi(c) = (\mathcal{Q}(c)', (\mathcal{Q}(c))'')$

Exemple : Soient le treillis initial de la Figure 2.8 et le treillis obtenu après insertion du nouvel objet ({9},{cdfgh}). $\chi^+(c_{\#6}) = c_{\#2}$. $\chi(c_{\#10}) = c_{\#7}$.

La Figure 2.9 récapitule toutes les applications définies dans ce cadre théorique.

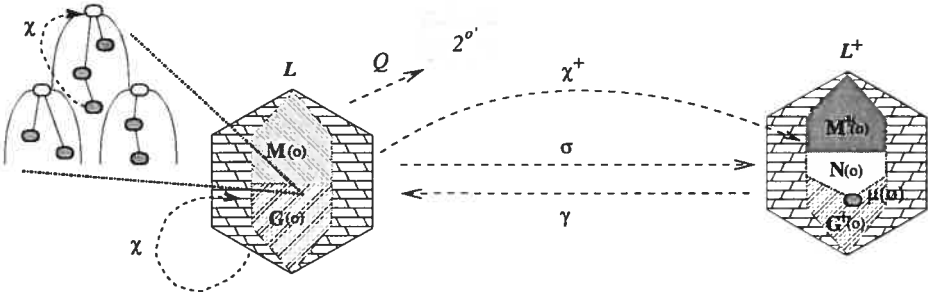


FIG. 2.9 – Les différentes applications reliant le treillis initial $\mathcal{L}$ au treillis $\mathcal{L}^+$ obtenu par l'ajout d'un nouvel objet o.

Si la classe d'un géniteur c_2 appartient à la couverture supérieure d'une autre classe ayant c_1 comme géniteur, alors il existe un concept de la couverture supérieure de c_1 qui appartient à $[c_2]_{\mathcal{Q}}$.

Propriété 11. *:* $\forall c_1, c_2 \in \mathbf{G}(o) \times \mathbf{G}(o) : [c_1]_{\mathcal{Q}} \leq_{/\mathcal{Q}} [c_2]_{\mathcal{Q}} \Rightarrow \exists \bar{c} \in [c_2]_{\mathcal{Q}} : c_1 \prec \bar{c}$.

Par conséquent, on peut déterminer les couvertures supérieures d'un nouveau concept en considérant seulement la couverture supérieure de son géniteur dans $\mathcal{L}$ et en prenant les minimas de leurs images par χ^+.

Propriété 12. *:* $\forall c, \bar{c} \in \mathbf{N}^+(o) \times \mathcal{C}^+ : c \prec^+ \bar{c} \iff \bar{c} \in min(\{\chi(\hat{c}) | \gamma(c) \prec \hat{c}\})$

Il est évident qu'un nouveau concept appartient toujours à la couverture supérieure de son géniteur. La propriété suivante montre que le seul concept de $\sigma(\mathcal{C})$ qui est couvert par un nouveau concept dans $\mathcal{L}^+$ est l'homologue de son géniteur.

Propriété 13. *:* $\forall \bar{c} \in \mathcal{C}^+ \setminus \mathbf{N}^+(o),\ \bar{c} \leq^+ c \iff \bar{c} \leq^+ \gamma(\sigma(c))$.

La propriété ci-dessous récapitule le calcul des couvertures supérieures dans le treillis $\mathcal{L}^+$:

Propriété 14. *La relation* $\prec^+$ *est obtenue à partir de* $\prec$ *de la manière suivante :*

$$\begin{array}{rl} \prec^+= & \{(\sigma(\gamma(c)), c) \mid c \in \mathbf{N}^+(o)\} \\ \cup & \{(c, \bar{c}) \mid c \in \mathbf{N}^+(o),\ \bar{c} \in \min(\{\chi(\hat{c}) \mid \gamma(c) \prec \hat{c}\})\} \\ \cup & \{(c_1, c_2) \mid (\gamma(c_1), \gamma(c_2)) \in (\prec - \mathbf{G}(o) \times \mathbf{M}(o))\} \end{array}$$

Cela signifie que l'ordre dans $\mathcal{L}^+$ est obtenu de la manière suivante :

- Pour un nouveau concept, son géniteur est un prédécesseur immédiat. Tous ses autres prédécesseurs sont des nouveaux concepts,
- Les successeurs immédiats d'un nouveaux concepts sont des concepts nouveaux ou modifiés,
- Les prédécesseurs immédiats d'un concept modifié c de $\mathcal{L}^+$ sont différents de ceux du concept $\gamma(c)$ par, d'une part, l'ajout d'un ensemble de nouveaux concepts et d'autre part la suppression de tous les concepts appartenant à $\mathbf{G}^+(o)$.

2.2.1.3 Description de l'algorithme

La méthode (Algorithme 1) que nous avons proposé dans [86] permet de mettre à jour un treillis suite à l'ajout d'un nouvel objet. Les opérations de mise à jour consistent à : (i) identifier les classes d'équivalence, (ii) trouver l'élément maxima de chaque classe $[]_{\mathcal{Q}}$ et de déterminer son statut (géniteur ou modifié), (iii) mettre à jour des concepts modifiés, (iv) créer les nouveaux concepts, (iv) déterminer les nouvelles couvertures inférieures et supérieures des nouveaux concepts, (v) et finalement éliminer les liens obsolètes de chaque générateur. Ces opérations sont réalisées de la manière suivante :

- **Détection des maxima de classes d'équivalence** $[]_{\mathcal{Q}}$: Comme seulement l'élément maxima est utilisé dans la restructuration du treillis initial, il est inutile de construire toute la classe d'équivalence. En effet, selon la propriété 15, pour tout concept c non maximal dans sa classe d'équivalence, il existe toujours un successeur $\bar{c}$ appartenant la classe $[c]_{\mathcal{Q}}$. Par conséquent, le statut d'un concept peut être déterminé en le comparant avec ses successeurs qui sont visités préalablement dans le cas d'un parcours descendant du treillis.

  Propriété 15 (détection des maxima des classes). *Toutes les classes* $[]_{\mathcal{Q}}$ *dans* $\mathcal{L}$ *sont des ensembles convexes :* $\forall\ c,\ \bar{c},\ \underline{c} \in \mathcal{C},\ \underline{c} \leq c \leq \bar{c}$ *et* $[\bar{c}]_{\mathcal{Q}} = [\underline{c}]_{\mathcal{Q}} \Rightarrow [\bar{c}]_{\mathcal{Q}} = [c]_{\mathcal{Q}}$.
- **Calcul des couvertures supérieures des nouveaux concepts :** De plus, pour le calcul des couvertures supérieures d'un nouveau concept c, au lieu de considérer tous les successeurs potentiels, on peut prendre l'ensemble $\{\chi^+(\bar{c}) \mid \bar{c} \in Cov^u(\gamma(c))\}$, c'est à dire, les minima des images par χ^+ de tous les successeurs du géniteur $\gamma(c)$.

  Propriété 16. *Pour tout* $c = (X, Y) \in \mathcal{L}$, *et* $\bar{c}_1 = (\bar{X}_1, \bar{Y}_1), \bar{c}_2 = (\bar{X}_2, \bar{Y}_2) \in Cov^u(c)$, $\bar{X}_1 \cap \bar{X}_2 = X$.
- **Destruction des liens obsolètes :** Enfin tout modifié $\hat{c}$ qui est un successeur immédiat d'un générateur $\bar{c}$ dans $\mathcal{L}$ doit être déconnecté de $\bar{c}$ dans $\mathcal{L}^+$ car $\chi^+(hatc)$ est nécessairement un successeur du nouveau concept produit par $\chi^+(barc)$.

  Propriété 17. *Pour tout* $\bar{c} \in \mathbf{G}(o), \hat{c} \in \mathbf{M}(o) : \bar{c} \prec \hat{c} \Rightarrow \hat{c} \in \min(\{\chi^+(\hat{c}) \mid \hat{c} \in Cov^u(\bar{c})\})$.

L'algorithme 1 prend un treillis $\mathcal{L}$ est un nouvel objet[10] et produit le treillis à jour $\mathcal{L}^+$. Les valeurs de *calQ* et χ^+ sont représentées par une structure générique (la structure *ChiPlus*) indexée par les identificateurs de concepts. Au début, les concepts sont triés par ordre croissant de la taille de l'intension afin de permettre un parcours descendant du treillis (la routine SORT de la ligne 3). Le traitement itératif

[10]L'ensemble A est supposé être connu, c'est à dire, $\{o\}' \subseteq A$.

```
1: procedure ADD-OBJECT(In/Out : L = ⟨C, ≤⟩ a lattice; In : o an object)
2:
3: SORT(C)
4: for all c in C do
5:    new-max ← ARGMAX({|Q(c̄)| | c̄ ∈ Cov^u(c)})
6:    if |Q(c)| ≠ |Q(new-max)| then
7:       if |Q(c)| = |Intent(c)| then
8:          Extent(c) ← Extent(c) ∪ {o}      {c is modified}
9:          M(o) ← M(o) ∪ {c}
10:         new-max ← c
11:      else
12:         ĉ ← NEW-CONCEPT(Extent(c) ∪ {o},Q(c))      {c is genitor}
13:         Candidates ← {ChiPlus(c̄) | c̄ ∈ Cov^u(c)}
14:         for all c̄ in MIN-CLOSED(Candidates) do
15:            NEW-LINK(ĉ,c̄)
16:            if c̄ ∈ M(o) then
17:               DROP-LINK(c, c̄)
18:         new-max ← ĉ
19:         L ← L ∪ {ĉ}
20:   ChiPlus(c) ← new-max
```

Algorithm 1: Mise à jour du treillis suite à l'ajout d'un nouvel objet au contexte.

(lignes 4 jusqu'à ligne 20) examine chaque concept c du treillis $\mathcal{L}$ et détermine son statut dans $[c]_{\mathcal{Q}}$ en comparant $|\mathcal{Q}(c)|$ à la valeur maximale de $|\mathcal{Q}(\bar{c})|$ où $\bar{c}$ est un successeur immédiat de c (ligne 6). Pour ce faire, la variable *new-max* est utilisée. Cette dernière désigne le concept dans $\mathcal{L}^+$ dont l'intension est égale à $\mathcal{Q}(c)$, c'est à dire, $\chi^+(c)$. Elle est initialisée avec le successeur $\bar{c}$ ayant la valeur maximale $|\mathcal{Q}(\bar{c})|$ (ligne 5). Les maxima de classes sont répartis en modifiés et géniteurs (ligne 7). Pour un concept modifié c (lignes 8 to 10), l'extension est mise à jour par l'objet o et est considérées comme étant son propre maxima (ligne 10) par l'intermédiaire de la variable *new-max* ($\chi^+(c) = c$). Un géniteur donne, d'abord naissance à un nouveau concept $\hat{c}$ (ligne 12) ensuite la valeur de χ^+ de chaque successeur immédiat est mise dans une liste locale (la liste *Candidates*, ligne 13) afin de pouvoir sélectionner les éléments minima (MIN-CLOSED, ligne 14). Les concepts minima sont liés au nouveau concept $\hat{c}$ (ligne 15) et ceux qui sont des modifiés dans $\mathcal{L}$ (ligne 16) sont déconnectés du géniteur c (ligne 17). Finalement, le maxima de

$c \in \mathcal{L}$	$\mathcal{Q}(c)$	*new-max*	Statut	$c \in \mathcal{L}^+$	$\chi^+(c)$
$c_{\#1}$	$\emptyset$	*null*	M(9)	$\{c_{\#1}\}$	$c_{\#1}$
$c_{\#2}$	c	*null*	M(9)	$\{c_{\#2}\}$	$c_{\#2}$
$c_{\#3}$	d	*null*	M(9)	$\{c_{\#3}\}$	$c_{\#3}$
$c_{\#4}$	g	*null*	M(9)	$\{c_{\#4}\}$	$c_{\#4}$
$c_{\#5}$	h	*null*	M(9)	$\{c_{\#5}\}$	$c_{\#5}$
$c_{\#6}$	c	$c_{\#2}$	U(9)	$\{c_{\#6}\}$	$c_{\#2}$
$c_{\#7}$	cd	*null*	G(9)	$\{c_{\#7}, c_{\#14}\}$	$c_{\#14}$
$c_{\#8}$	dgh	*null*	M(9)	$\{c_{\#8}\}$	$c_{\#8}$
$c_{\#9}$	fh	*null*	G(9)	$\{c_{\#9}, c_{\#15}\}$	$c_{\#9}$
$c_{\#10}$	cd	$c_{\#7}$	U(9)	$\{c_{\#10}\}$	$c_{\#10}$
$c_{\#11}$	$cdgh$	*null*	G(9)	$\{c_{\#11}, c_{\#16}\}$	$c_{\#11}$
$c_{\#12}$	$dfgh$	*null*	G(9)	$\{c_{\#12}, c_{\#17}\}$	$c_{\#12}$
$c_{\#13}$	$cdfgh$	*null*	G(9)	$\{c_{\#13}, c_{\#18}\}$	$c_{\#13}$

TAB. 2.4 – Trace de l'exécution de l'algorithme 1 avec le treillis de la Figure 2.7 et le nouvel objet ($\{9\}$,$\{cdfgh\}$).

la classe $[c]_{\mathcal{Q}}$ dans $\mathcal{L}^+$, c'est à dire, $\chi^+(c)$ est repositionné (ligne 18), le nouveau concept est insérer dans l'ensemble des concepts du treillis (ligne 19) et la structure *ChiPlus* qui représente χ^+ est mise à jour par le nouveau maxima de la classe $[c]_{\mathcal{Q}}$ pour une future utilisation (ligne 20).

Exemple : Soient le contexte de la Table 2.3 formé par $\mathcal{K} = (O = \{1, \ldots, 8\}, A = \{a, \ldots, h\}, I)$ et le contexte $\mathcal{K}^+$ obtenu par l'ajout de la paire ($\{9\}$,$\{cdfgh\}$). La Table 2.4 illustre l'exécution de l'algorithme 1. Le treillis initial est donné par la Figure 2.7 tandis que le treillis résultat est donné par la Figure 2.8.

Pour illustrer la manière avec laquelle l'algorithme 1 traite un concept donné, considérant le cas du concept $c_{\#12} = (3, defgh)$. La valeur de $\mathcal{Q}(c_{\#12})$ est $dfgh$. Comme les successeurs immédiats de $c_{\#12}$ sont $c_{\#8}$ et $c_{\#9}$ (voir Figure 2.7), la liste des candidats contenant les images par χ^+ de ces successeurs est égale à : *Candidates*= $\{\underline{c}_{\#8} = (139, dgh), \underline{c}_{\#15} = (359, fh)\}$. Il est évident qu'aucune intension de successeur n'est plus grande que celle de $\mathcal{Q}(c_{\#12})$. Ainsi, $c_{\#12}$ est un maxima dont le statut est un un géniteur. Le nouveau concept $\underline{c}_{\#17} = (39, dfgh)$ est créé et connecté à tous ses successeurs immédiats dans *Candidates* car ils sont tous des

éléments incomparables. Finalement, comme $c_{\#8}$ est un modifié, le lien avec $c_{\#12}$ est supprimé.

2.2.1.4 Complexité

Soient $l = \|\mathcal{C}_\mathcal{K}^+\|$, $m = \|A\|$, $k = \|O\|$ et $\Delta l = \|\mathcal{C}_\mathcal{K}^+\| - \|\mathcal{C}_\mathcal{K}\|$. Le tri des concepts (ligne 3) est linéaire dans la taille du treillis car, en effet, il s'agit d'une comparaison entre les tailles des intensions qui sont bornées par m. Pour la boucle (ligne 4-20), nous avons un facteur de l. Le calcul de $\mathcal{Q}(c)$ est borné par m, car il s'agit d'intersection d'intension. La couverture supérieure d'un concept $c = (X, Y)$ contient un nombre de concepts qui est borné par $n - \|X\|$, en effet un concept $\bar{c} = (\bar{X}, \bar{Y})$ est un successeur de c si $\bar{X} = X \cup \bar{O}$ tel que $\bar{O} \subseteq O \setminus X$ et ainsi le calcul de la complexité de l'opération ARGMAX appartient à $O(m+n)$. En ce qui concerne les concepts modifiés, les lignes (8-10) sont linéaires en fonction de la taille de $\mathbf{M}(o)$ et peuvent être négligées. L'autre opération coûteuse est la partie liée aux concepts géniteurs $\mathbf{G}(o)$. En effet, le nombre de géniteurs est Δl. La mise à jour de l'ordre, effectuée pour chaque géniteur, a un coût de $O(n^2)$ qui peut s'expliquer par le parcours de la couverture supérieure dont on a vu ci-dessus que sa taille est en $O(n)$ et pour chaque élément de la couverture, un calcul d'intersection sur les extensions également en $O(n)$. Les autres opérations sont à temps constant. Donc, la partie liée aux géniteurs a un coût global en $O(\Delta l \cdot n^2)$. Le coût total de l'ajout d'un objet est en $O(l \cdot (m+n) + \Delta l \cdot n^2)$. Enfin, la complexité de l'algorithme pour l'ajout de tous les objets est de l'ordre de $O(nl \cdot (m+n))$.

2.2.2 Incrémentalité par attribut

Une manière de construire le treillis $\mathcal{L}$ consiste à ajouter, progressivement, un attribut a_i au treillis $\mathcal{L}_{i-1}$ (qui correspond au contexte $\mathcal{K} = (O, \{a_1, ..., a_{i-1}\}, I)$), en commençant par le seul attribut a_1 et en faisant à chaque étape une série de mises à jour structurelles. Ce type d'incrémentalité à fait l'objet de notre contribution et est présenté dans le Chapitre 5, Section 5.3.1.

2.2.3 Outils de construction/visualisation de treillis

Plusieurs programmes sont disponibles permettant la construction de treillis. Le plus ancien (mais aussi parmi les plus fiable) d'entre eux est CONIMP qui s'exécute sous le système MSDOS et est accessible à l'adresse http ://www.mathematik.tu-darmstadt.de/ags/ag1/Software. Un autre programme plus moderne développé par *Navicon Company* pour les applications d'analyse de données est *Navicon Decision Suite* (TOSCANA, ANACONDA, CERNATO). Une version démo est accessible à l'adresse : http ://www.navicon.de. CONEXP est aussi un outil qui implémente toutes les fonctionnalités de base dans le domaines de l'AFC est disponible à l'adresse : http ://sourceforge.net/projects/conexp/. TOSCANAJ est aussi un explorateur de schémas conceptuels et optimisé pour des novices en AFC et est disponible à l'adresse : http ://toscanaj.sourceforge.net/.

Pour nos propres expérimentations, nous avons entrepris un projet d'une grande envergure visant la mise au point d'une plateforme à source ouverte, baptisée GALICIA [84, 85] qui supporte la totalité du cycle de vie d'un treillis (voir Chapitre 7, Section 9.3). Une distribution gratuite de cette plateforme est accessible à l'adresse : https ://sourceforge.net/projects/galicia/.

2.3 Quelques applications usuelles de l'AFC

L'AFC est appliquée dans de nombreux domaines tels que l'analyse de données, le génie logiciel, la recherche d'information, l'extraction de connaissances, etc. Ces applications ont amplement contribué au développement théorique et algorithmique de l'AFC. Nous allons décrire quelques applications où les recherches sont très actives en ce moment. Une bonne partie de cette section est consacrée à la description de la manière avec laquelle l'AFC a abordé les problèmes du génie logiciel en particulier la réingénierie des systèmes et les solutions proposées pour améliorer leur qualité.

2.3.1 AFC et extraction de connaissances

L'extraction de connaissances[11] [61] à partir de données (ECD) est une activité qui consiste à analyser les données afin de dégager de manière non triviale des informations implicites. À l'heure actuelle, ce domaine de recherche se trouve à la croisée de chemins de nombreux autres domaines tels que : systèmes de bases de données, intelligence artificielle, bases de données spatiales, visualisation de données, etc. Le processus d'ECD débute par des données brutes et abouti à des connaissances qui sont potentiellement utiles, dépendamment du type de données, à la gestion, le contrôle, la conception, la commercialisation, etc. Un processus classique d'ECD comprend le nettoyage et l'intégration de données à partir de diverses sources, la sélection et au besoin la transformation des données, la fouille de données[12] et finalement l'évaluation et la présentation des connaissances extraites. L'étape cruciale dans ce processus est la fouille de données dont le but et de chercher les patrons informatifs à partir des données et de les présenter sous une forme souhaitée telle que des règles d'implications ou une classification. C'est à ce moment qu'intervient l'AFC. En effet, l'AFC permet de représenter les données brutes par le biais de contextes formels et de dériver la hiérarchie conceptuelle correspondante [21]. Des règles d'implication [30] entre attributs peuvent être alors déduites de manière directe à partir du treillis permettant ainsi l'induction des hypothèses sur les relations entre attributs, ou d'extraire une classification des objets. L'avantage de l'utilisation de l'AFC avec le calcul des règles d'implication est la garantie d'obtenir une base de règles minimale (sans redondance des règles). De plus, les travaux sur l'incrémentalité des treillis ont conduit au développement d'algorithmes efficaces de mise à jour incrémentale d'une base de règles suite à l'évolution des données correspondantes.

[11] En Anglais knowledge discovery.
[12] En Anglais data-mining.

2.3.1.1 Règles d'association et bases de règles

Le calcul des règles d'association [1] est considéré comme l'un des types de base d'extraction de connaissances dans les bases de données. Étant donnée une base de données, calculer les règles d'association consiste à déterminer toutes les règles qui ont un support minimum et une confidence. Une règle d'association est une implication de la forme $X \Rightarrow Y$, où X et Y sont des sous-ensembles de A, appelés aussi "*motifs*", et $X \cap Y = \emptyset$. Le support d'une règle $X \Rightarrow Y$ est défini comme étant $supp(X \cup Y)$ alors que sa confidence est le rapport $supp(X \cup Y)/supp(X)$.

L'étape cruciale dans le calcul des règles d'association est la détermination des patrons fréquents à partir desquels les règles sont générées. Le nombre de motifs fréquents peut être potentiellement grand conduisant à un nombre élevé de règles. Une approche basée sur les opérateurs de fermeture a été proposée permettant de réduire le nombre des règles et par conséquent leur interprétation.

Ainsi, le calcul des patrons fréquent a été ramené au calcul des motifs fermés fréquents[13] qui caractérise l'information pertinente d'un motif fréquent et qui conduit finalement à un ensemble de règles d'association minimal et représentatif, appelé "base". Plusieurs types de bases de règles d'association ont été proposés dans la littérature telles que la base de Duquenne-Guigues, base de couverture de Luxemburger, base générique et la base informative. Ces bases sont produites à partir du treillis iceberg.

Les règles approximatives de couverture de Luxemburger sont générées à partir du motif fermé fréquent du concept et celui de ses prédécesseurs immédiats. La base de Duquenne-Guigues nécessite le calcul des pseudo-intensions fréquentes associées au motif fermé fréquent d'un concept fréquent du treillis. La génération des règles exactes à partir de la base de Duquenne-Guigues est triviale car la pseudo-intension devient une prémisse et le motif fermé fréquent du concept devient la conséquence de la règle.

Dans les bases génériques et informatives, les *générateurs* d'un motif fermé

[13]En Anglais Frequent Closed Itemsets (FCI).

fréquent joue un rôle crucial dans le calcul des règles. À noter qu'un générateur Z d'un ensemble fermé X est le sous-ensemble minimal de X tel que $Z'' = X$. Les règles exactes de la base générique sont produites à partir des concepts fréquents en utilisant leurs générateurs et les motifs fermés fréquents correspondants. Les règles approximatives de la base informatique sont produites à partir des générateurs du concept fréquent et du motif fermé fréquent de ses prédécesseurs immédiats.

2.3.2 AFC et recherche d'information

De nombreux travaux combinant AFC et recherche d'information ont montré le bien fondé de l'utilisation de l'AFC en recherche d'information [12]. La correspondance entre les éléments de base de l'AFC et de la recherche d'information a été vite établie. En effet, les individus du contexte sont les objets recherchés (documents, utilisateurs, etc.) et les propriétés sont les caractéristiques qui leurs sont communes (mots-clés, liens hyper-documents, profiles, etc.). Un concept formel dans ce contexte représente un cluster d'individus ayant les mêmes propriétés, par exemple un ensemble de documents partageant un ensemble de mots-clés.

L'AFC permet de concilier les deux modes de recherche d'information : les systèmes hiérarchiques et les systèmes booléens. En effet, le treillis, grâce à la factorisation maximale des propriétés, offre la structure de recherche la plus minimale qui combine les deux formes de recherche, à savoir, la navigation entre sous-population d'individus et l'interrogation (requête) de sous-ensembles de propriétés. L'exploration de la structure est garantie par les liens conceptuels offerts par le treillis.

Le principe d'exploration de la hiérarchie conceptuelle établie par les méthodes de l'AFC consiste à partir d'une requête vide qu'on raffine par ajouts successifs d'attributs (mots-clés). Chaque étape, on avance davantage dans la structure ce qui permet de restreindre l'espace de recherche, jusqu'à arriver à satisfaire la requête. On a donc une recherche par navigation dont le point de départ est le concept maximal du treillis (tous les objets) et le point d'arrivée est le concept possédant

tous les attributs de la requête. L'extension de ce concept est rendue comme résultat de la recherche.

2.3.3 AFC et réingénierie du logiciel

Les systèmes légataires [7, 49], devenus de plus en plus obsolètes dans leur architecture et/ou plateforme, présentent des difficultés à supporter l'évolution en fonction des changements de besoins. L'intérêt de la réingénierie [2, 45, 15] est l'amélioration ou la transformation de ces systèmes afin de les rendre plus compréhensibles, fiables, maintenables, assez documentés et facile à faire évoluer ou à migrer vers un autre paradigme. Les systèmes orientés objets (OO) contemporains peuvent aussi être soumis à une activité de réingénierie visant à corriger la conception ou à dégager un ensemble d'artéfacts potentiellement utiles à la maintenance. La réingénierie consiste donc à prendre un système existant et le reconstruire sous une autre forme en se basant sur un ensemble de processus élémentaires telles que la rétroingénierie, la redocumentation, la restructuration, etc. La rétroingénierie est le processus de compréhension et d'analyse d'un système dans le but d'identifier les composantes de celui-ci ainsi que leurs relations et de créer des représentations (abstractions) de ce système sous différentes formes ou à différents niveaux d'abstraction.

Depuis que Wille et Ganter ont développé l'AFC pour en faire une technique d'analyse et de dérivation de structures conceptuelles [30, 95], les applications ont proliféré, notamment en génie logiciel. En effet, grâce à sa faculté de regroupement et de structuration hiérarchique, l'AFC s'applique à des problèmes de la réingénierie [2, 45, 15] de logiciels où l'on s'intéresse à analyser le code existant pour des fins de compréhension et de maintenance.

Snelting [76, 77] et Godin [32, 35] ont présenté une vue générale des applications de l'AFC en réingénierie du logiciel, notamment la réingénierie des configurations logicielles [75], la dérivation et l'évaluation de la structure modulaire des systèmes légataires [51, 74], l'identification des objets dans le code procédurale [50, 73, 93]

et la réingénierie des hiérarchies de classes [32, 35, 77].

Dans ce qui suit, nous allons décrire la manière avec laquelle l'AFC a abordé les problèmes de la réingénierie de logiciels orientés objet et les solutions proposées pour améliorer la qualité des systèmes.

2.3.3.1 Réingénierie des systèmes OO

De nos jours, on parle aussi de système légataires orientés objet qui souffrent eux aussi des mêmes symptômes que les systèmes légataires procéduraux. Des techniques de restructuration "refactoring" sont appliquées pour garder ces systèmes opérationnels. Cette restructuration est principalement dûe aux anomalies suivantes [26] :

- usage impropre de l'héritage : la réutilisation du code versus polymorphisme,
- manque d'héritage : la duplication du code ou présence de l'instruction "switch",
- opérations mal placées : des opérations hors classes appropriées,
- violation de l'encapsulation : le transtypage "casting" d'instances et méthodes "friends",

Ainsi, pour remédier à ces anomalies dans la conception objet et à l'instar des systèmes procéduraux, les systèmes à objet ont largement bénéficié de l'apport de l'AFC comme une technique d'analyse et de structuration. En effet, plusieurs aspects de la conception objet ont été traités tels que la construction d'une hiérarchie de classes initiale à partir de la spécification des classes ou des objets, réingénierie des hiérarchies de classes existantes à partir des relations entre les classes et leurs attributs/méthodes membres, évolution de la hiérarchie de classes à partir des nouveaux besoins et l'assemblage de hiérarchies existantes.

2.3.3.1.1 Conception et réingénierie des hiérarchies de classes : Une activité majeure dans l'élaboration du modèle objet consiste à identifier les classes et les relations sémantiques les reliant. Les relations d'héritage représentent les liens sémantiques les plus sensibles du fait de l'influence majeure qu'elle ont sur

la compréhensibilité des modèles, l'intégrité sémantique des modèles, le potentiel de réutilisation des classes résultantes pour d'autres applications, et l'effort de développement requis.

Le problème de construction d'une hiérarchie de classes de départ à partir des spécifications des différentes classes ou bien la réorganisation d'une hiérarchie existante a reçu une attention particulière dans la littérature de l'OO. Du côté de l'AFC, le contexte formel est composé de classes et de membres de classes qui représentent, respectivement, les objets et attributs formels. La relation d'incidence, dans ce cas, représente pour chaque variable les membres contenus dans son type. Le treillis obtenu à partir de ce contexte comporte beaucoup de redondance. Ainsi, ce treillis subit plusieurs transformations qui consistent à éliminer les éléments redondants pour arriver à une structure qui n'est plus un treillis mais qui demeure un ordre partiel. Cette structure est la hiérarchie[14] de classes recherchée.

2.3.3.1.2 Travaux antérieurs : Godin et Mili [32, 35] ont utilisé les treillis de Galois pour la réorganisation de la hiérarchie d'héritage en se basant sur la signatures des classes. Le point de départ dans leur approche est un ensemble d'interfaces de classes. Une table binaire est construite représentant pour chaque interface l'ensemble des méthodes supportées. Le treillis dérivé à partir de cette table montre comment la hiérarchie de classes implémentant ces interfaces doit être organisée afin d'optimiser la répartition des méthodes dans la hiérarchie. Les travaux de Godin et Mili ont les mêmes bases formelles que ceux de Snelting [77], toutefois Godin et Mili considèrent les relations entre les membres et les classes alors que Snelting a étudié comment les membres d'une hiérarchie de classes sont utilisés dans le code exécutable de plusieurs applications en examinant non seulement les relations entre les classes et les membres de classes mais aussi les relations entres les différents membres de classes. Huchard et *al.* [41] ont proposé une méthode de restructuration d'un modèle de classes UML permettant de considérer aussi bien les classes que les associations dans l'élaboration des généralisations.

[14]Une hiérarchie de classes a seulement besoin d'être un ordre partiel.

2.3.4 Discussion

Toute discipline dont les données peuvent être codées en un contexte formel (objets x attributs) peut bénéficier des avantages de l'AFC. En effet, l'AFC de part ses capacités de conceptualisation, structuration et factorisation apportent une solution élégante à chaque fois que les données brutes doivent être regroupées (concepts) selon des critères de similarité bien établis (partage objets/attributs) ou encore classifiées sans redondances (factorisation maximale). Toutefois, la taille des structures conceptuelles (treillis, iceberg, etc.) produites par les techniques de l'AFC posent un obstacle à l'exploitation de ces structures car elle dépend de manière exponentielle du volume de données. Ceci affaiblit la position de l'AFC face aux disciplines dont les ensembles de données (datasets) sont de grande taille.

2.4 Conclusion

Bien que l'AFC a été, initialement, introduite comme une alternative à la théorie des treillis [30], la discipline a amplement outrepassé ce cadre étroit. En effet, divers domaines, tels que les sciences sociales, l'analyse de données, la découverte de connaissances, le génie logiciel, etc., ont adopté le cadre de l'AFC et ont fait de ses méthodes des outils efficaces pour l'extraction de structures conceptuelle à partir de données brutes. Toutefois, l'AFC présente certaines limites devant un nombre croissant d'applications ayant besoin de représenter et traiter des informations complexes qui outrepassent le paradigme classique de la relation d'incidence. Exemple de telles informations sont les liens qui peuvent surgir entre les individus au sein d'un même contexte ou en provenance de divers contextes. Ces informations sont très importantes car une fois considérées dans le processus de dérivation des structures conceptuelles peuvent conduire à la découverte d'abstractions utiles.

Chapitre 3

AFC et les représentations structurées

Dans ce chapitre, nous allons présenter quelques extensions de l'AFC qui ont été proposées dans le but de traiter des données complexes où la description des individus ne peut se faire via le paradigme classique de la relation binaire en AFC. Nous allons aussi présenter le langage de modélisation UML. Toutefois, le lien entre l'AFC et UML, qui représente l'intérêt central de cette thèse, est laissée au Chapitre 6.

3.1 AFC et la logique

Dans cette section nous allons rappeler les notions fondamentales des logiques de descriptions. Nous allons aussi étudier les similarités entre ces logiques et l'AFC de point de vue de la représentation de connaissances et de l'inférence. À la fin, nous discutons de quelques extensions logiques de l'AFC qui lui permettent de traiter des contextes plus complexes où la description des individus ne peut se résumer en la présence/absence de propriétés (attributs binaires).

3.1.1 Logiques de descriptions

Les logiques de descriptions sont des langages de représentation de connaissances qui constituent des sous-ensembles de la logique des prédicats [3]. Ces langages sont plus expressifs que les langages de Frames et les réseaux sémantiques et permettent de représenter les connaissances terminologiques d'une manière structurée et de faire des inférences.

3.1.1.1 Représentation des connaissances

Étant donné un domaine, les individus représentent les objets de ce domaine. Les concepts représentent des classes d'individus et les relations expriment les liens inter-individus. Représenter les connaissances du domaine en LD consiste à exprimer de façon bien définie les concepts et les relations entre eux. Ainsi, les logiques de descriptions focalisent les concepts et leurs définitions en termes de relations.

En effet, la base de connaissances est définie à partir de deux ensembles de symboles, à savoir, les noms de concepts[1] ($\mathcal{T}_C$) et les noms de rôles[2] ($\mathcal{T}_R$). Ces concepts et rôles sont dits *primitifs* et sont comparables, respectivement, aux prédicats unaires et binaires de la logique des prédicats.

Exemple : Dans l'exemple des publications scientifiques sur l'application de l'AFC aux activités du génie logiciel (Table 4.1), les concepts considérés sont AboutDetailedDesign, AboutMaintenance, AboutArchitecture et AboutRequirements tandis que le rôle cites (voir la Table 4.2 pour les liens entre publications) exprime la relation de citation entre les publications.

3.1.1.2 langage de description des concepts

La représentation de connaissances en LD s'appuie sur un *langage de concepts* où les descriptions primitives sont combinées pour exprimer des descriptions plus complexes en utilisant des constructeurs tels que : conjonction ($\sqcap$), disjonction ($\sqcup$),

[1]Le nom d'un concept commence toujours par une majuscule.

[2]Le nom d'un rôle commence toujours par une minuscule.

quantification universelle ($\forall$), quantification existentielle ($\exists$), définition de concepts $\equiv$, etc. De plus, le concept universel, appelé `top` ($\top$), et le concept absurde, appelé `bottom` ($\bot$) peuvent aussi être utilisés.

Exemple : La description de toutes les publications sur la conception détaillée citant uniquement des publications sur la maintenance est comme suit :

$$\mathsf{AboutDetailedDesign} \sqcap \forall \mathsf{cites.AboutMaintenance}$$

La sémantique des descriptions dans la base de connaissances est donnée par le biais d'une interprétation $\mathcal{I} = (\Delta^{\mathcal{I}}, .^{\mathcal{I}})$ définie de la manière suivante :

Définition 37. *Une interprétation $\mathcal{I}$ est une paire $(\Delta^{\mathcal{I}}, .^{\mathcal{I}})$, qui consiste en un ensemble non vide d'individus $\Delta^{\mathcal{I}}$ appelé domaine d'interprétation et une fonction interprétation $.^{\mathcal{I}}$ qui associe à chaque nom de concept, C, un sous-ensemble $C^{\mathcal{I}} \subset 2^{\Delta^{\mathcal{I}}}$ et à chaque nom de rôle, r, un sous-ensemble de couples $r^{\mathcal{I}} \subset 2^{\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}}$.*

Ainsi, la sémantique des descriptions est de nature extensionnelle, c'est à dire, spécifiée par la définition de l'ensemble des individus dénotés par chaque description de concept ou de rôle. Par exemple, l'intersection de concepts $C \sqcap D$ permet de restreindre l'ensemble des individus à ceux appartenant à C et D simultanément $(C^{\mathcal{I}} \cap D^{\mathcal{I}})$. De manière similaire, l'interprétation de la restriction de rôle $\forall r.C$ est l'ensemble des individus ayant une relation r avec seulement des individus appartenant à $C^{\mathcal{I}}$. La Table 3.1 (adaptée de [3]) récapitule la syntaxe et la sémantique des expressions d'un langage de description de la famille $\mathcal{FL}_0$.

Exemple : L'interprétation de la description ci-dessus par rapport au contexte des publications (Table 4.1) et la relation de citation (Table 4.2) est :

$$\{1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 23, 24, 25\}.$$

3.1.1.2.1 Le langage $\mathcal{FL}_0$: Le choix d'un sous ensemble de constructeurs détermine le langage LD et a un impact direct sur la complexité des mécanismes de raisonnement. Une large gamme de langages LD ont été développés allant du plus simple (pas de disjonction, pas de négation) au plus expressif telle que la

Nom du constructeur	Syntaxe	Sémantique
nom de concept	C	$C^{\mathcal{I}} \subset \Delta^{\mathcal{I}}$
concept top	$\top$	$\Delta^{\mathcal{I}}$
concept bottom	$\bot$	$\emptyset$
conjonction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
quantification universel	$\forall r.C$	$\{x \in \Delta^{\mathcal{I}} \mid \forall y : (x,y) \in r^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$
définition de concept	$C \equiv D$	$C^{\mathcal{I}} = D^{\mathcal{I}}$
collection d'individus	$\{a_1, \ldots, a_n\}$	$\{a_1^{\mathcal{I}}, \ldots, a_n^{\mathcal{I}}\}$

TAB. 3.1 – Syntaxe et sémantique des constructeurs de formation de concepts en $\mathcal{FL}_0$.

famille $\mathcal{AL}$ [3]. Par exemple, l'ensemble $\{\sqcap, \equiv, \forall, \texttt{top}, \bot\}$ constitue le langage $\mathcal{FL}_0$. Des langages plus puissants sont définis en rajoutant d'autres constructeurs.

Définition 38. *Étant donné $\mathcal{T}_C$, $\mathcal{T}_R$, deux ensembles finis de noms de concepts et de noms de rôles, respectivement. $\mathcal{FL}_0(\mathcal{T}_C, \mathcal{T}_R)$ désigne l'ensemble des descriptions de concepts inductivement formés comme suit :*

- $\mathcal{T}_C \cup \{\top, \bot\} \subseteq \mathcal{FL}_0$,
- $C, D \in \mathcal{FL}_0 \Rightarrow C \sqcap D \in \mathcal{FL}_0$,
- $C \in \mathcal{FL}_0, R \in \mathcal{T}_R \Rightarrow \forall R.C \in \mathcal{FL}_0$.

3.1.1.3 Système à base de connaissances

Un système à base de connaissances s'appuyant sur les LD est caractérisé par deux composantes principales, à savoir, la `TBox`, qui définit le domaine en terme de concepts et de relations (le modèle), et la `ABox`, qui définit les individus et les liens inter-individus (les données). La `ABox` est une instanciation de la `TBox`.

3.1.1.3.1 Terminologie (`TBox`) : La définition $A \equiv C$ permet d'associer à une description de concept C un nom de concept A. Ainsi, des noms ("macros") qui correspondent à des descriptions de concepts ou de rôles peuvent être définis à partir de la paire ($\mathcal{T}_C$, $\mathcal{T}_R$). L'ensemble de ces définitions est appelé `TBox` où chaque

nom est défini une et une seule fois[3].

Un langage LD est défini de manière inductive sur une TBox $\mathcal{T}(\mathcal{T}_C, \mathcal{T}_R)$. Par exemple, le langage $\mathcal{FL}_0(\mathcal{T})$ est la fermeture de l'application des opérateurs $\sqcap$ et $\forall$ aux concepts dans $\mathcal{T}_C \cup \{\top, \bot\}$ et les rôles dans $\mathcal{T}_R$.

De notre part, nous considérons les TBox dans un cadre général, c'est à dire, même le cas ou elles contiennent des cycles terminologiques, c'est à dire, des définitions de concepts qui font une référence directe ou indirecte aux noms de concepts définis. À noter que les cycles terminologiques posent des problèmes lors du calcul des relations de subsomption et d'équivalence [3, 54].

Alternativement, si $\mathcal{T}$ est une TBox à partir de $(\mathcal{T}_C, \mathcal{T}_R)$ (possiblement cyclique), c'est à dire, un ensemble de définitions $\{A \equiv C | A \in \mathcal{T}_C, C \in \mathcal{FL}_0(\mathcal{T}_C, \mathcal{T}_R)\}$, le langage induit par la TBox $\mathcal{T}$ est $\mathcal{FL}_0(\mathcal{T})$.

Exemple : Les déclarations suivantes, exprimées dans le langage KRSS[4], font partie de la TBox associée à l'exemple des publications de la Table 4.1 :

```
(implies P4 (and P2 P5))
(implies P6 (and sm P5))
(implies P10 (and P8 (all cites P7)))
```

3.1.1.3.2 Description du domaine (ABox) : Une base de connaissances comprend aussi une ABox $\mathcal{A}$ où des assertions sur les individus du domaine représenté sont stockées.

Exemple : Les assertions suivantes en langage KRSS figurent dans la ABox représentant le domaine des publications de la Table 4.1 :

```
(instance Fun95 aboutDetailedDesign)
(instance Kro94 aboutSoftwareMaintenance)
(related Fun95 Kro94 cites)
```

[3]La circularité dans les définitions n'est pas permise, toutefois cette contrainte est ignorée dans notre cas.

[4]Description-Logic Knowledge Representation System Specication, www.dl.kr.org/krss-spec.ps.

3.1.1.4 Services d'inférence

Il y a plusieurs service d'inférence qu'un raisonneur sur une base de connaissances LD peut effectuer tels que :

- Equivalence : Deux descriptions de concepts C et D sont équivalentes, notée $C \equiv D$, si $C^{\mathcal{I}} = D^{\mathcal{I}}$ pour toutes les interprétations $\mathcal{I}$. La vérification de l'équivalence sert à déterminer si une nouvelle description existe déjà dans la base.
- Classification : consiste à affecter les individus aux différents concepts.
- Subsomption : la relation de *subsomption* est le service inférentiel de base fourni par un raisonneur LD. Elle consiste à déterminer si un concept C est un sous-concept du concept D. D'un point de vue sémantique, le concept C est plus général (≪subsumes ≫) que le concept D, noté $D \sqsubseteq C$, si $D^{\mathcal{I}} \subseteq C^{\mathcal{I}}$. Le calcul de la relation de subsomption permet de construire une hiérarchie de concepts de la `TBox` facilitant ainsi la recherche et la réutilisation des concepts.
- Satisfiabilité : consiste à vérifier s'il existe une interprétation $\mathcal{I}$ telle que $C^{\mathcal{I}} \neq \emptyset$ ($\mathcal{I}$ est appelé un modèle de C). Ce service permet de tester si un concept peut être instanciable (contenir des individus). Il faut noter que la subsomption peut être réduite à une satisfiabilité.
- Classification : Le calcul de la hiérarchie de subsomption (taxonomie) de tous les concepts nommés.

3.1.1.5 AFC v.s. LD

Du point de vue représentation de connaissances, AFC et LD s'entendent sur la modélisation des concepts du domaine. Toutefois, chacune des deux disciplines met l'emphase sur un élément différent. En effet, l'AFC focalise sur le mécanisme par lequel les concepts émergent à partir d'observations individuelles et par conséquent elle est de nature inductive. De plus, en AFC les concepts ont une sémantique aussi bien extensionnelle qu'intensionnelle. Dans la dimension intensionnelle, un concept

est décrit par le biais d'une conjonction de descripteurs (attributs formels) binaires[5]. Pour leur part, les logiques de descriptions se focalisent sur l'expressivité dans la représentation des connaissances sur les concepts et les relations inter-concepts ainsi que le raisonnement déductif comme moyen d'inférence sur ces connaissances, tandis que les observations individuelles ont un rôle secondaire.

Par conséquent, AFC et LD sont complémentaires et la mise en place d'une intégration étroite entre ces deux disciplines est profitable pour de nombreuses applications, notamment, la structuration automatique des bases de connaissances [64, 4, 71].

Nous allons décrire dans le Chapitre 4, section 4.8, la manière d'exporter les résultats d'un cadre étendu de l'AFC en une base de connaissances LD. Le lecteur peut, par conséquent, observer les correspondances entre les éléments du cadre fondamental de l'AFC et ceux des logiques de descriptions.

3.1.2 Extensions logiques de l'AFC

L'application de l'AFC à certains domaines a révélé les limites du schéma binaire (présence/absence d'attributs) offert par le cadre fondamental de l'AFC pour la description des objets. Par conséquent, plusieurs extensions de ce cadre ont été proposées pour permettre aux contextes d'encoder des entités ayant une description complexe.

Nous nous intéressons particulièrement aux extensions qui ont procédé à la généralisation du langage de description des objets, à savoir l'extension de Chaudron [13] où les propriétés élémentaires des objets sont remplacées par des expressions de la logique des prédicats et l'extension de Ferré [25] où les ensembles d'attributs sont remplacés par des expressions de la logique. Nous avons aussi étudié l'extension de Prediger [62] qui utilise le mécanisme du scaling dans la dérivation des descriptions binaires à partir de descriptions complexes en utilisant une terminologie définie. Ce même mécanisme est utilisé dans l'extension que nous proposons.

[5]Qui sont dans certains cas le résultat d'un scaling complexe

3.1.2.1 Extension aux termes de la logique du premier ordre

L'étude de Chaudron et Maille [13] s'inscrit dans le cadre du développement d'un module de génération de règles du premier ordre pour le système TC1 [6] destiné à la découverte de connaissances dans l'interaction homme-machine. Le système a été utilisé pour l'analyse des bases de connaissances contenant des facteurs humains de pilotage d'un avion.

Dans le développement de TC1, Chaudron et Maille ont introduit la logique des prédicats en AFC. Les propriétés élémentaires des objets sont remplacées par des expressions de la logique des prédicats. Ainsi, les objets sont décrits par des conjonctions de littéraux positifs appelés *cubes logiques*. L'ensemble de tous les littéraux est appelé "*domaine des cubes*", noté $\mathcal{C}$. Le modèle des cubes est basé sur la logique du premier ordre (*Constante, Variable, Fonction, Prédicat et Quantificateurs*). Un contexte du premier ordre $\mathcal{K} = (O, P, \square)$ est composé de l'ensemble classique des objets O, d'un ensemble de cubes P, avec $P \subseteq 2^{\mathcal{C}}$, généré à partir d'un langage de la logique du premier ordre et d'une application $\square : O \rightarrow P$ qui retourne, pour chaque objet $o \in O$, une et une seule image dans P représentant l'ensemble des traits qui lui correspondent ($p = \square(o)$).

Le domaine des cubes logiques $\mathcal{C}$ et muni d'une opération $\bigcap_{\mathcal{C}}$ permettant de capturer les traits communs de deux cubes, et une opération $\bigcup_{\mathcal{C}}$ pour réunir les traits de deux cubes. Ces opérations sont simplement la généralisation des opérations usuelles d'intersection $\bigcap$ et d'union $\bigcup$ sur les ensembles d'attributs.

Les correspondances de Galois f et g qui résument les liaisons entre sous-ensembles d'objets et des cubes sont définies de la manière suivante : la fonction f associe à un ensemble d'objets l'unification de leur cubes respectifs, tandis que g est la fonction duale pour les littéraux :

- pour un ensemble fini d'objets X, $f : \mathcal{P}(O) \rightarrow P$, $f(X) = \bigcap_{\mathcal{C}}\{\square(o) | o \in X\}$
- pour un littéral donné Y, $g : P \rightarrow \mathcal{P}(O)$, $g(Y) = \{o \in O | Y \leq_{\mathcal{C}} \square(o)\}$

La relation d'ordre $\leq_{\mathcal{C}}$ est la relation induite par la structure du treillis sur

[6] Un système Prolog pour l'analyse de concepts de premier ordre.

C. Un concept du premier ordre est une paire (X, Y) telle que : $X \subset O$, $Y \in P$, $f(X) = Y$.

Pour deux concepts donnés (X_1, Y_1) et (X_2, Y_2), la relation hiérarchique de *prédécesseur - successeur* est formalisée par : $(X_1, Y_1) \leq (X_2, Y_2) \Leftrightarrow X_1 \subseteq X_2$ ($Y_2 \leq_C Y_1$). L'ensemble de tous les concepts du premier ordre du contexte $(O, P, \square)$, muni de la relation d'ordre $\leq_C$ (la relation d'ordre induite par la structure du treillis sur C) est un treillis complet, appelé "*treillis de premier ordre*".

Enfin notons que Chaudron et Maille ont proposé le remplacement des ensembles d'attributs par des cubes logiques ce qui permet d'étendre l'AFC à une logique particulière. Cette extension, motivée par un besoin d'expressivité pour des applications en analyse de données, manque de généricité [7], car il faut redéfinir le paradigme de l'AFC pour d'autres applications ayant besoin d'autres logiques.

3.1.2.2 Généralisation logique de l'AFC

Ferré, dans [25] présente un paradigme logico-contextuel pour interroger, naviguer et apprendre dans un système d'information, appelé "Système d'information logique". Ce paradigme se distingue des modèles existants par une organisation des données centrée sur les objets (fichiers, pages Web) plutôt que sur les propriétés (répertoires) ; l'ensemble des objets décrits par leurs propriétés constitue le contexte. Le paradigme effectue une recherche d'information qui combine étroitement l'interrogation et la navigation. Les propriétés des objets sont exprimées par un langage logique. En effet, Ferré propose une reformulation de l'analyse de concept classique d'une manière à pouvoir faciliter la définition d'une analyse de concept généralisée à toute logique, appelée 'Analyse de Concepts Logiques' (ACL), dans laquellle le langage de description des objets devient un paramètre. Cette généricité dans la nature des objets et la logique choisie, peut couvrir beaucoup de domaines d'application.

La reformulation du cadre classique de l'AFC consiste à remplacer le contexte

[7] Chaudron et Maille ont publié, par la suite, d'autres généralisations de l'AFC en restant toujours dans le cadre de la logique des cubes

et les correspondances de Galois de la manière suivante :

$$\mathcal{K} = (O, A, I) \Longrightarrow \mathcal{K} = (O, 2^A, d), \text{ où } d(o) = \{a \in A | (o, a) \in I\}.$$

2^A est le domaine de description des objets et d est la fonction de description des objets dans ce domaine. Ensuite, les correspondances de Galois sont reformulées avec la fonction d plutôt que la relation I de la manière suivante :

$$f(X) = \bigcap_{o \in X} d(o) \text{ et } g(A) = \{o \in O | d(o) \supseteq A\}.$$

Une logique $\mathcal{L} = (L, \sqsubseteq, \sqcap, \sqcup, \top, \bot)$ est définie pour l'analyse de concepts logiques permettant de substituer les ensembles d'attributs et les opérations ensemblistes par des formules et des opérations logiques (voir Table 3.2).

AFC	ACL		description
2^A	L	–	langage des formules logiques
$\supseteq$	$\sqsubseteq$	$\sqsubseteq \subseteq L \times L$	relation de subsomption
$\cup$	$\sqcap$	$\sqcap \in L \times L \Longrightarrow L$	l'opération de conjonction
$\cap$	$\sqcup$	$\sqcup \in L \times L \Longrightarrow L$	l'opération de disjonction
$\emptyset$	$\top$	$\top \in L$	la formule distinguée tautologie
A	$\bot$	$\bot \in L$	la formule distinguée contradiction
I	d	–	la fonction de description des objets

TAB. 3.2 – Correspondance entre type abstrait logique de l'ACL et l'AFC.

La généralisation est obtenue en remplaçant les sous-ensembles d'attributs 2^A par un ensemble de formules de $\mathcal{L}$. Finalement, le contexte formel, appelé *contexte logique* est un triplet $(O, \mathcal{L}, d)$ où :

- O est un ensemble fini d'objets,
- $\mathcal{L} = \langle L, \sqsubseteq \rangle$ est un treillis de formules,
- $d : O \mapsto \mathcal{L}$ une fonction qui associe à chaque objet une formule décrivant ses propriétés.

Dans le contexte $(O, \mathcal{L}, d)$, un concept est une paire $c = (X, Y)$ où $X \subseteq O$, $Y \in \mathcal{L}$, tel que $f(X) = Y$ et $g(Y) = X$. La différence principale par rapport à l'AFC est que l'intension Y est une formule dans une logique $\mathcal{L}$. De plus, une relation de déduction contextuelle est introduite dans certains contextes particuliers. Cette

relation est considérée comme une généralisation de l'implication entre attributs. La logique contextuelle (la logique des déductions du contexte formel) joue le même rôle que le treillis de concepts et permet d'extraire la connaissance à partir des contextes.

Ferré propose trois façons pour construire le treillis : des extensions ($g(f(2^O))$) ordonnées par inclusion ($\subseteq$), des intensions $f(g(\mathcal{L}))$) ordonnées par déduction logique, ou des formules ($\mathcal{L}^K$) ordonnées par une déduction contextuelle.

3.1.2.3 Extension aux variables non binaires

Pour remédier à l'impossibilité de traiter de manière directe les données non binaires (par exemple, une base de données relationnelle où la relation d'incidence est de type objet-attribut-valeur) pour en extraire des hiérarchies conceptuelles, Prediger [62, 63] a proposé deux méthodes permettant de dériver un contexte formel mono-valué à partir d'un contexte pluri-valués : les échelles conceptuelles et le scaling logique.

3.1.2.3.1 Échelles conceptuelles Les échelles conceptuelles (conceptual scales) sont des contextes formels dont les objets sont des valeurs et les attributs sont des attributs d'échelle. Ces contextes induisent des treillis de concepts qui organisent d'une manière hiérarchique les attributs de l'échelle. chaque attribut du contexte pluri-valué, on associe une telle échelle conceptuelle qui permet de structurer le domaine de valeurs de cet attribut (voir Section 2.1.3 pour plus de détails).

3.1.2.3.2 Scaling logique Le principe de scaling logique est d'utiliser un langage formel (ex., SQL) pour générer des prédicats unaires à partir des attributs et des valeurs d'un contexte pluri-valué. Une terminologie est un ensemble fini de tels prédicats auxquels on associe un nom. Le contexte mono-valué est dérivé du contexte pluri-valué à partir d'une terminologie. L'ensemble d'objets est conservé, l'ensemble d'attributs est l'ensemble des noms des prédicats de la terminologie et un objet est en relation avec un nom si les valeurs de ses attributs sont tels que le

prédicat associé au nom est satisfait.

Par exemple, si la terminologie contient la définition $assez\ petit =_{def} taille \leq 3$ (assez petit est le nom du prédicat et taille est un attribut) et que la valeur de l'attribut *taille* d'un objet est 2, alors cet objet aura l'attribut *assez petit* dans le contexte dérivé.

Ce que l'on reproche à ces deux type de scaling est le fait qu'il peut y avoir perte d'information. En effet, avec le premier scaling les valeurs sont remplacées par des attributs d'échelle, ce qui correspond à une forme d'abstraction de données. Avec le deuxième type de scaling, on s'intéresse à la satisfaction ou non de quelques prédicats (exprimés dans un langage formel) par les objets. Toutefois, pour des applications d'analyse de données, cela peut être utile puisqu'il s'agit de discerner les propriétés générales des données.

3.1.3 Extensions relationnelles de l'AFC

Diverses recherches ont tenté d'étendre l'AFC pour lui permettre de traiter les objets dont les descriptions incluent des liens. Un lien est une référence à une partie de l'objet (lien intra-objet) ou à un autre objet du contexte (lien inter-objets).

3.1.3.1 Traitement des liens intra-objet

Pour le traitement des liens intra-objet, différentes approches ont été proposées en se basant sur des langages de prédicats [13] ou encore des graphes conceptuels [53]. En effet, les descriptions des objets incluant des liens intra-objet ressemblent à des prédicats reliant des parties d'objet entre elles et peuvent être, par conséquent, représentées par des graphes où les noeuds sont ces parties et les arcs sont ces prédicats (voir à gauche de la Figure 3.1). Ainsi, l'extraction de concepts formels nécessite une méthode particulière de calcul basée sur l'isomorphisme de graphes.

L'approche basée sur les graphes ainsi que les autres approches basées sur la logique du premier ordre considèrent exclusivement les liens intra-objet, c'est à dire,

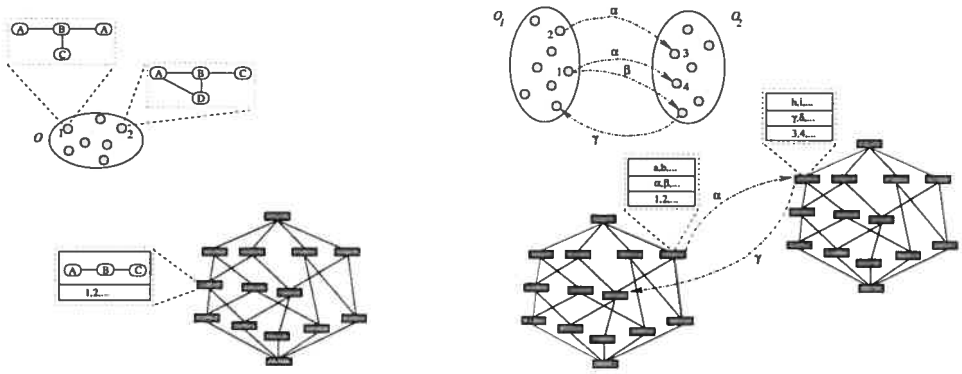


FIG. 3.1 – **Gauche** : traitement des liens intra-objets. **Droit** : traitement des liens inter-objets.

des liens qui ne dépassent jamais la 'frontière' de objet formel concerné. Ainsi, les liens sont internes aux descriptions des concepts. Par contre, dans notre cas les liens inter-objets font partie des intensions des concepts découverts.

3.1.3.2 Extension aux liens inter-objets

L'analyse relationnelle proposée par U. Priss [65] est une autre extension de l'AFC vers une théorie plus générale permettant d'étendre les liens entre objets à des relations entre concepts. Elle a été, initialement, introduite pour résoudre le problème d'implémentation des relations sémantiques dans les bases de données lexicales.

Un contexte linguistique (dictionnaire, base de données lexicale, etc.) est formalisé par un contexte lexical formel. Deux types de contextes lexicaux sont définis : le contexte dénotatif et le contexte connotatif où les objets formels sont respectivement les mots et les sens. Dans les deux contextes, les propriétés sont des caractéristiques définies à partir de la terminologie linguistique.

L'approche consiste à construire le treillis de concepts (treillis linguistique) indépendamment des liens inter-objets (relation sémantique et lexicale entre mots) ; ensuite lever ces liens aux niveaux des concepts, c'est à dire établir des relations entre concepts à base des liens inter-objets. Des termes lexicaux et expressions

peuvent être dérivés des treillis linguistiques.

Cependant, il est toujours nécessaire d'examiner si la relation d'un concept donnée est valable pour tous les objets et attributs de l'extension et l'intension de ce concept ou seulement pour un sous ensemble. Par conséquent, le problème majeur de la propagation des liens inter-objets aux relations entre concepts est la quantification. Pour résoudre ce problème, des quantificateurs du langage naturel {*tous*, *exactement un*, *au moins un*} sont associés aux relations et permettent de décrire les conditions dans lesquelles les relations sont établies. Ainsi, établir une relation entre deux concepts consiste à définir la *composante relationnelle* (nom de relation) et les quantificateurs affectés aux extensions des deux concepts. Ces relations sont déduites à partir des relations sémantiques et lexicales (synonymie, taxonomie, etc.).

La démarche proposée par Priss pour la construction des treillis relationnels consiste à définir des relations entre les concepts une fois le treillis construit (traitement "à posteriori") en utilisant les liens inter-objets.

3.1.4 Discussion

Afin de permettre à l'AFC de répondre à certains besoins où la description classique (conjonction de propriétés) des objets présentent des limites, plusieurs extensions ont été proposées notamment celles qui introduisent les logiques classiques dans le langage de description des objets [13, 25, 62]. Pour notre part, pour pouvoir décrire des objets ayant des liens avec d'autres objets, nous avons procédé à une extension du cadre fondamental de l'AFC vers un cadre plus général où les liens inter-objets sont élevés au niveau des concepts. Par rapport à l'approche de Priss [65] qui consiste à établir les relations inter-concepts à posteriori (après la formation des concepts), notre approche utilise les liens inter-objets comme un critère de formation de concepts. Le chapitre 4 en présentera les détails.

3.2 Le langage de modélisation UML

Dans cette section nous allons décrire le modèle structurel de UML, à savoir le modèle de classes. Notre intérêt dans ce modèle est motivé par l'application que nous allons réaliser et qui consiste en la restructuration d'une hiérarchie de classes représentée par un diagramme UML. Nous allons aussi présenter le format standard d'échange de méta-données dit "*XML Metadata Interchange*" (XMI)[8].

3.3 Le langage

Le langage de modélisation unifié UML [60, 27] est un langage de spécification, visualisation, construction et documentation des systèmes logiciels. C'est un langage de modélisation par objets, devenu un standard depuis qu'il a été adopté par l'OMG (Object Management Group) en 1997. Ses concepts proviennent des notations présentées par Grady Booch [9], James Rumbaugh [72] et Ivar Jacobson [43].

À noter que UML n'est pas une simple notation graphique ni une méthode dans la mesure où elle ne présente aucune "démarche". A ce titre elle est un formalisme de modélisation objet qui fournit des éléments syntaxiques pour la plupart des systèmes logiciels, ce qu'UML désigne sous le nom d'"artéfact". UML a l'ambition d'être au génie logiciel ce que la notation symbolique est aux mathématiques. Le mot méthode est parfois utilisé par abus de langage.

Les principaux objectifs de la notation UML sont :

- offrir aux utilisateurs un langage de modélisation visuel pour développer et communiquer des modèles,
- fournir des mécanismes de spécialisation pour l'extension des concepts fondamentaux (langage ouvert),
- supporter des spécifications indépendamment du processus de développement ou du langage de programmation[9],

[8]http ://www.omg.org/technology/documents/formal/xmi.htm

[9]UML est aussi un langage de choix dans le cadre de l'architecture MDA (Model Driven Architecture). En effet, UML est utilisée au niveau de l'organisation comme un PIM (Plateform

- fournir une base formelle pour la compréhension d'un langage de modélisation et intégrer des pratiques de modélisation élégantes,
- encourager la croissance du marché d'outils de l'objet,
- supporter des concepts de développement de haut niveau tels que les composants et les collaborations.

UML constitue un langage formel et normalisé ce qui entraîne un gain de précision, un gage de stabilité et encourage l'utilisation d'outils. Son caractère polyvalent et sa souplesse en font de lui un support de communication performant et universel qui cadre l'analyse et facilite la compréhension de représentations abstraites complexes.

UML couvre toutes les phases d'un projet logiciel et propose une modélisation suivant trois axes : comportemental, structurel et architectural. La modélisation comportementale est le processus consistant à attribuer des responsabilités aux classes qui composent le système. Il s'agirait de l'activité la plus importante au cours du cycle de vie, car le comportement dicte la structure du système. En effet, modéliser le comportement permet de déterminer les responsabilités de la classe ainsi que les interactions avec d'autres objets au moment de l'instanciation. La modélisation structurelle consiste à identifier les classes d'objets, les relations et de généralisation/spécialisation entre classes et les types d'associations entre classes. La modélisation comportementale et la modélisation structurelle sont plus efficaces lorsqu'elles sont menées en parallèle. Finalement, la modélisation de l'architecture du système logiciel consiste à décrire l'organisation des composants logiciels, les moyens d'interconnexion et les ressources matérielles nécessaires à l'implantation du logiciel. Chaque axe de modélisation dispose d'un ensemble de diagrammes parmi les suivants : diagramme de classes, objets, cas d'utilisation, activité, état-transition, collaboration, séquence, composants et déploiement.

Ainsi, le langage UML fournit une panoplie d'outils permettant de représenter l'ensemble des éléments du monde objet (classes, objets, etc.) et les liens qui les relient. Toutefois, étant donné qu'une seule représentation est trop subjec-

Independent Model).

tive, UML fournit un moyen permettant de représenter diverses projections d'une même représentation grâce aux *vues*. Une vue est une projection du modèle et est constituée de plusieurs diagrammes. Dans le cadre de notre travail, nous nous intéressons à la vue qui permet de représenter les aspects statiques et structurels d'un système, à savoir la vue *logique* et plus particulièrement au diagramme de classes et aux diagramme d'objets. Dans la suite, nous allons présenter les constituants essentiels de ce diagramme.

3.3.0.1 Diagramme de classes

Le diagramme de classes décrit la structure statique d'un système à travers un ensemble d'éléments de modélisation, tels que les classes, les interfaces, et les relations qui les relient. Il fait abstraction des aspects dynamiques et temporels du système. Pour un modèle complexe, plusieurs diagrammes de classes complémentaires peuvent être construits. Les composants suivants constituent les éléments d'un diagramme de classes UML.

3.3.0.1.1 Classe une classe est la description d'une collection d'objets qui partagent les mêmes propriétés (attributs, opérations, relations) et la même sémantique. Graphiquement, une classe est représentée par un rectangle avec trois compartiments : compartiment classe incluant le nom de la classe, son stéréotype et ses valeurs étiquetées, compartiment des attributs[10] et enfin celui des méthodes[11].

La classe est aussi caractérisée par une visibilité ou encore droits d'accès. Trois niveaux de droits d'accès sont disponibles : privée, protégée, publique et package qui spécifient la portée de la classe par rapport aux différents packages du système. UML permet de préciser le nombre d'instances d'une classe par le biais de la multiplicité de la classe.

[10]Pour un attribut donné, il faut préciser le nom, le type de base et la visibilité.

[11]Pour une opération, il faut préciser la signature et la visibilité.

3.3.0.1.2 Relations les relations représentent les liens entre les classes. UML offre trois types de relations : la généralisation/spécialisation, les associations et les dépendances.

3.3.0.1.3 Association une association exprime une connexion sémantique bi-directionnelle entre deux classes. Les associations permettent de naviguer dans le modèle (voyager d'un objet à l'autre). En effet, l'association est instanciée dans un diagramme d'objets ou de collaboration, sous forme de liens entre objets issus de classes associées. L'association possède un nom qui décrit la nature de la relation et un rôle qui précise la façon dont une classe se présente à une autre (indispensable pour les associations réflexives). Les multiplicités d'une association précisent le nombre d'instances qui participent à une relation. Graphiquement, une association est représentée sur le modèle par une droite reliant deux classes. Les rôles et multiplicités des deux côtés, le sens de navigation, les contraintes de l'association figurent aux 'bords' de cette droite. On distingue plusieurs types d'associations :

- *Agrégation* association binaire entre un tout et ses parties. L'agrégation implique que le tout se situe à un niveau conceptuellement plus élevé que la partie, tandis qu'une association implique que les deux classes se trouvent au même niveau conceptuel.
- *Composition* : une forme spéciale d'agrégation dans laquelle la partie ne peut survivre sans son tout.
- *Classe d'association* : UML permet de modéliser les propriétés d'une association entre deux classes par une classe d'association. Il s'agit d'une classe qui réalise la navigation entre les instances d'autres classes.
- *Qualification* : un moyen pour limiter le nombre d'objets en cas de navigation d'associations car la connaissance de l'objet source et du qualificateur permet de limiter le nombre d'objets liés.

3.3.0.1.4 La dépendance exprime une relation d'utilisation unidirectionnelle. Une modification de la classe dont on dépend peut nécessiter une mise à jour de la

classe dépendante. Souvent les dépendances seront utilisées pour démontrer qu'une classe en utilise une autre comme paramètre dans la signature d'une opération.

3.3.0.1.5 La généralisation consiste à factoriser les propriétés d'un ensemble de classes sous forme d'une superclasse, plus abstraite. La spécialisation est la relation duale qui consiste à étendre les propriétés d'une classe, sous forme de sous-classes, plus spécifiques. Les deux relations permettent de définir une structure hiérarchique pour un diagramme de classes. La généralisation est l'élément UML utilisé pour modéliser l'héritage.

3.3.0.1.6 Exemple : le diagramme de classes de la Figure 3.2, tiré de [32], décrit les éléments membres d'une agence immobilière. La classe `Locataire` a deux attributs : le `nom` du `Locataire` et son `adresse`. La classe `Maison` est caractérisée seulement par son `type`. Deux associations apparentes dans le contexte immobilier sont `acheter` et `louer` qui correspondent à la relation entre la `Maison` et ses deux principaux acteurs, à savoir le `Propriétaire` et le `Locataire`. Comme une `Maison` ne peut être occupée que par un seul `Locataire`, la cardinalité de l'association `Louer` est égale à "`0..1`" du coté du `Locataire`. Par contre un `Locataire` peut `Louer` plusieurs `Maison`. Cette règle est exprimée par la même association avec la cardinalité "`*`" qui se trouve du coté de la classe `Maison`. Le sens de la navigation de l'association `Louer` est du `Locataire` vers la `Maison`. Cela signifie qu'à partir d'un `Locataire` donné, il est toujours possible de déterminer la `Maison` qu'il occupe. Par contre il est impossible de déterminer l'occupant d'une `Maison` donnée.

3.3.1 Le Méta-modèle

UML est un langage permettant d'exprimer des modèles objet en faisant abstraction de leur implémentation. Ce langage s'appuie sur un méta-modèle, un modèle de plus haut niveau qui définit les éléments d'UML (les concepts utilisables) et leur sémantique (leur signification et leur mode d'utilisation). Le méta-modèle d'UML est un langage formel, sans ambigüité et pouvant servir de support pour

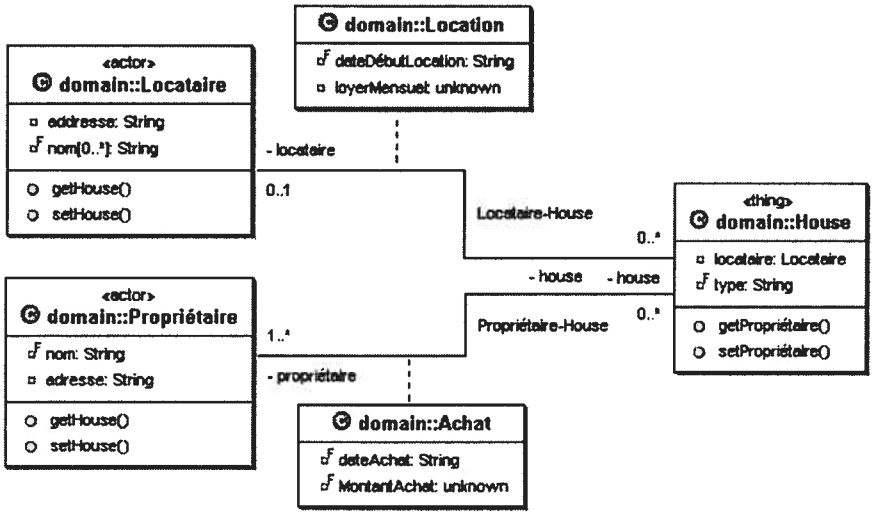


FIG. 3.2 – Transactions immobilières : diagramme de classes UML.

tout langage orienté objet. Il est décrit par la notation UML.

Le méta-modèle de UML est écrit en langage standard de métamodélisation appelé MOF (Meta-Object Facility). MOF est un ensemble d'interfaces standards permettant de définir et de modifier des méta-modèles et leurs modèles correspondants (UML, Java, OMG-IDL, méta-modèle workflow WfMF, etc.) en tant qu'objets CORBA. CORBA est un concept permettant à deux applications de développer des échanges d'informations à l'abri de leur localisation ou de leur conception interne. Le MOF est donc un moyen de définir la syntaxe et la sémantique d'un langage de modélisation. Il a été créé par l'OMG afin de définir la notation UML mais peut aussi remplir d'autres responsabilités éventuellement établir les correspondances entre méta-modèles. Par exemple le méta-modèle de mapping entre UML et Java est écrit en MOF.

3.3.2 Modélisation avec UML

Comme souligné ci-haut, le langage UML répond au besoin de spécifier, représenter et construire les composantes d'une future application objet, indépendamment d'un

langage de programmation. En effet, le langage a apporté un formalisme et une cohérence aux applications à développer. Toutefois, malgré la richesse d'expression de UML, il est toujours difficile de construire des modèles dont la qualité de la conception avoisine celle préconisée par l'approche objet, en particulier quand il s'agit du modèle conceptuel de classes. Rappelons qu'il y a plusieurs critères de qualité pour un modèle de classes tels que la factorisation maximale des propriétés des classes, l'utilisation de l'héritage multiple seulement si nécessaire, le nombre de classes limité, sous-classer dans le but de spécialiser, etc. Ce manque de qualité est due, d'une part au manque d'une démarche systématique de construction du modèle et d'autre part à la complexité que présente cette activité. Cette complexité découle principalement d'un espace de recherche d'abstractions très grand, des critères parfois conflictuels, évolution permanente en rapport avec la réalité, etc.

Comme nous allons le voir dans le chapitre 8.2.2.2, l'utilisation de l'AFC a contribué à la lever de certaines difficultés, telle que l'organisation d'un ensemble de classes en une hiérarchie, en offrant une technique formelle de regroupement. En effet, l'AFC a été employée avec succès dans la construction et la restructuration des hiérarchies de classes du fait des propriétés intrinsèques des treillis qui garantissent une structure minimale et une factorisation maximale de la hiérarchie dérivée des données extraites du code source. Cette structure est soumise à des simplifications permettant d'éliminer les redondances et la ramener à un schéma d'héritage.

3.3.3 UML et AFC : limites et défis

Dans le chapitre 2, section 2.3.3, nous avons discuté la longue tradition d'utilisation de techniques basées sur l'AFC en génie logiciel. Toutefois, il reste encore des questions auxquelles il n'y pas encore de réponse ni de la part du paradigme classique de l'AFC, ni de la part de ses diverses extensions. Pour soulever ces questions, nous allons considérer un problème en génie logiciel où l'AFC peut servir à découvrir des abstractions (sous-classes, sous-types) en s'appuyant sur des res-

trictions de valeurs d'objets. Dans ce cas, chaque classe (respectivement entité) est traduite en un contexte formel où les objets formels correspondent aux instances de la classe (objets concrets) et les attributs formels correspondent aux propriétés communes de ces instances. Ainsi, le modèle de classes est décrit par un ensemble de contextes. Mais, en considérant les liens entre les objets (les attributs de référence dans la description des objets en OO ou les clés étrangères dans le schéma ER) les contextes formels deviennent liés et l'AFC est incapable, à ce moment, de traiter ce genre d'information.

Un autre exemple permettant aussi de mettre en évidence les limites du cadre classique de l'AFC relève du problème d'identification des objets dans le code procédural où le contexte est fait de routines (objets formels) et de variables (attributs formels). Cette perception du code conduit à la découverte de classes potentielles qui facilitent la migration du code procédural à la technologie objet. Mais, une telle modélisation ne capture pas toute l'information disponible dans le code source, notamment les appels inter-routines. Par conséquent, une partie de la sémantique est perdue empêchant ainsi la découverte d'abstractions utiles pouvant contribuer à une migration plus efficace.

L'application de l'AFC aux modèles conceptuels consiste aussi à manipuler des modèles déjà élaborés. Dans le cas d'un diagramme UML de classes, les objets formels correspondent aux classes du diagramme et les attributs formels aux différents membres de ces classes. L'AFC classique peut aider à produire une hiérarchie qui représente des abstractions obtenues à partir de généralisations sur les classes uniquement [32, 35], alors que les associations, qui représentent un aspect important du modèle, sont délaissées.

3.3.4 Discussion

Dans cette section nous avons présenté les éléments essentiels offerts par UML pour concevoir l'aspect statique des systèmes logiciels. Nous avons aussi exposé les difficultés rencontrées dans la construction des modèles conceptuels UML, notam-

ment les hiérarchies de classes. Ces difficultés conduisent souvent à des défauts[12] de conception, éventuellement, le manque de généricité ou de factorisation dans le modèle. L'AFC, grâce à ses techniques d'analyse et de structuration, permet la correction de ces défauts et l'amélioration conceptuelle d'un modèle UML (ajouter automatiquement de nouvelles classes/associations plus génériques). Toutefois, il reste encore des types d'information dans le modèle UML que l'AFC est incapable de traiter et qui nécessite une extension appropriée du cadre classique de l'AFC.

[12]Les défauts de conception peuvent se produire pendant la construction ou à la suite de la modification du modèle.

Chapitre 4

Fondements théoriques de l'ARC

Le but de ce chapitre est de présenter les fondements théoriques de l'analyse relationnelle de concepts (ARC). L'ARC est une approche permettant l'extraction de concepts formels à partir d'une description hétérogène des objets formels combinant des attributs formels binaires[1] et des relations inter-objets [16, 40]. Les concepts formés sont dits "concepts relationnels" car les intensions qu'ils renferment font référence à d'autres concepts.

Nous allons d'abord présenter le modèle de données permettant de représenter les informations relationnelles, appelé famille de contextes relationnels (FCR), ensuite décrire un cadre formel de transformation de ces données s'appuyant sur le mécanisme de scaling afin de pouvoir utiliser les méthodes fondamentales de l'AFC de dérivation de structures conceptuelles.

4.1 Problématique

Le cadre classique de l'AFC permet d'analyser des données appartenant à des entités d'un même type et ayant les mêmes caractéristiques alors que souvent les données à analyser sont issues de domaines disposant de plusieurs sortes d'entités,

[1]Étant donné un objet formel, le codage binaire consiste à associer une valeur booléenne vrai/faux pour indiquer la présence d'un attribut formel donné.

chacune ayant ses propres caractéristiques. De plus, ces entités peuvent avoir des liens de divers types entre elles. C'est le cas par exemple des bases de données qui contiennent plusieurs types d'entités stockés dans des tables, chacune a sa propre description (champs de la table) et des liens inter-entités (clés étrangères). Par conséquent, plusieurs travaux de recherche se sont intéressés à accommoder l'AFC à des données plus riches[2]. Les méthodes de formation de concepts utilisées s'appuient sur le mécanisme de scaling conceptuel [30] pour la traduction des attributs formels complexes en attributs binaires. Toutefois, les attributs formels qui expriment les liens entre les objets formels ont été traités de manière *ad-hoc* comme nous l'avons montré dans le Chapitre 3, Section 3.1.3.

De notre part, nous proposons une nouvelle méthode de traitement de ces liens inter-objets qui représentent en fait de l'information dite "*relationnelle*". À noter que cette information relationnelle est rencontrée aussi bien au niveau des données comme mentionné ci-dessus qu'au niveau des modèles qui décrivent ces données, particulièrement dans les modèles de données de type ER et UML. En effet, ces modèles sont composés de plusieurs types d'éléments et de plusieurs liens inter-éléments (voir Chapitre 8.2.2.2, section 6.2). Inclure les informations relationnelles dans le processus d'analyse permet aux outils et à ceux qui les utilisent de capturer davantage de similarités dans les données et donc plus de connaissance sur la source. Ainsi, la représentation des données est plus précise et l'interprétation sera plus correcte.

4.2 Les relations dans les données

Les contextes formels permettent de représenter n'importe quelle entité (objet formel) possédant des propriétés (attributs formels) tels que des humains, produits, véhicules, etc. Par exemple, la Table 4.1 présente la transposée[3] d'un contexte mono-valué, appelé "`publications`" (noté $\mathcal{K}_p$) où les objets formels représentent

[2]Entités décrites par des attributs de type numériques, taxonomiques ou encore des ensembles.
[3]Par mesure d'économie de l'espace.

des publications académiques traitant de l'application de l'AFC en génie logiciel. Les attributs formels représentent des activités constituant le cycle de développement du logiciel[4]. La Figure 4.1 illustre le treillis associé.

	(1) Fun95	(2) God93	(3) God95	(4) God98	(5) Huc99	(6) Huc02	(7) Kro94	(8) Kui00	(9) Leb99	(10) Lin95	(11) Lin97	(12) Sah97	(13) Sif97	(14) Sne96	(15) Sne98C	(16) Sne98R	(17) Sne99	(18) Sne00S	(19) Sne00U	(20) Str99	(21) Til03S	(22) Til03T	(23) Ton99	(24) Ton01	(25) Van98
ab	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
ca	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
cd	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
ma	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1

TAB. 4.1 – La transposée du contexte formel des publications académiques ($\mathcal{K}_p$). Les attributs formels sont : analyse des besoins (ab), conception architecturale (ca), conception détaillée (cd) et maintenance (ma).

Les entités ayant des descriptions différentes sont placées dans des contextes différents. En plus des caractéristiques propres, plusieurs sortes de liens peuvent être définies entres des objets formels appartenant à différents contextes (ou le même). Ces relations expriment les liens inter-objets dans le domaine correspondant tels que des liens de parenté, de possession ou seulement exprimer des informations techniques, tels que les hyperliens dans les bases de données textuelles ou dans le Web. La relation "possède", par exemple, peut représenter les liens entre des humains et des types particuliers de véhicules. Ainsi, dans une modélisation en AFC, "possède" exprime une relation entre le contexte des humains et celui des véhicules.

Concernant le contexte des publications de la Table 4.1, la Table 4.2 montre la relation circulaire de citation, appelée "cite", entre les différentes publications académiques. Ainsi, cette relation définit des liens entres les objets de la Table 4.1.

Les liens inter-objets représentent un aspect important lors d'un processus de construction de structures conceptuelles car ils peuvent servir à la découverte de

[4]Adapté de [81].

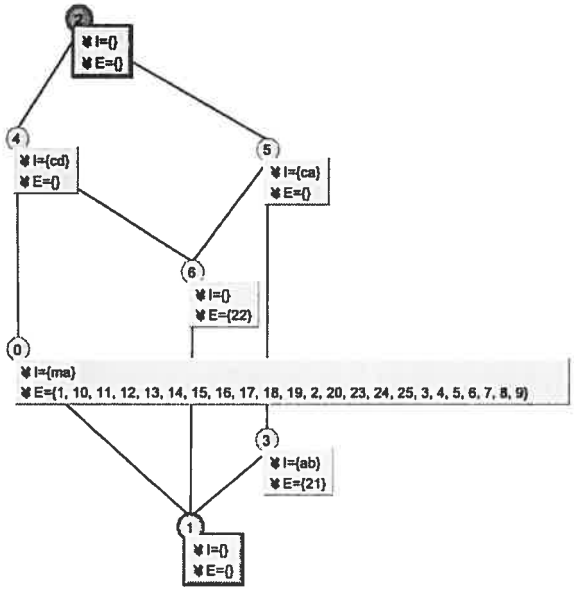


FIG. 4.1 – Le treillis initial $\mathcal{L}_p^0$.

	Fun95	God93	God95	God98	Huc99	Huc02	Kro94	Kui00	Leb99	Lin95	Lin97	Sah97	Sif97	Sne96	Sne98C	Sne98R	Sne99	Sne00S	Sne00U	Str99	Til03S	Til03T	Ton99	Tone01	Van98
Fun95	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
God93	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
God95	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
God98	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Huc99	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Huc02	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0
Kro94	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Kui00	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	0	1	0	0	0	0	0	0	1
Leb99	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Lin95	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Lin97	1	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
Sah97	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Sif97	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0
Sne96	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Sne98C	1	0	0	0	0	0	1	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0
Sne98R	0	1	0	0	0	0	1	0	0	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0
Sne99	1	1	0	1	0	0	1	0	0	0	1	0	1	1	1	1	0	0	0	0	0	0	0	0	0
Sne00S	0	0	0	0	0	0	1	0	0	0	1	0	1	1	0	1	1	0	0	0	0	0	0	0	1
Sne00U	0	1	0	1	0	0	1	0	0	0	1	0	0	1	1	1	0	0	0	0	0	0	0	0	0
Str99	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0
Til03S	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0
Til03T	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Ton99	0	0	0	0	0	0	1	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0
Tone01	0	0	0	0	0	0	1	0	0	0	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0
Van98	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	1	0	0	0	0	0	0	0	0	0

TAB. 4.2 – Relation de citation entre les différentes publications académiques.

groupe d'objets utiles à l'analyse. Par exemple, dans l'analyse d'un entrepôt de données sur les achats effectuées par des clients par le biais de cartes de crédits, il est très utile d'identifier le profil acheteur de chaque client non seulement en fonction de ses propres caractéristiques telles que salaire, âge, etc., mais aussi en fonction des caractéristiques des cartes de crédit qu'il détient telles que limite de crédit, taux d'intérêt, garantie additionnelle sur produits, etc. Ici, les objets sont regroupés en deux ensembles distincts, à savoir, l'ensemble des clients et celui des cartes de crédits. La relation "*détient*" établit un lien entre les objets de ces deux ensembles.

À noter que des travaux antérieurs [22] dans le domaine de l'analyse relationnelle ont révélé les difficultés de :

- Traiter des objets formels dont la structure ne peut être décrite de manière standard en AFC car elle comporte, par exemple, des rapports avec d'autres objets.
- Traiter des relations de type un-à-plusieurs[5] et résoudre les dépendances circulaires entre les objets.

Dans ce qui suit, nous allons présenter un cadre permettant la représentation des informations relationnelles et la formation de concepts dont les intensions représentent aussi bien le partage des attributs binaires entre objets formels que celui des liens avec d'autres objets formels.

4.3 Le modèle de données de base de l'ARC

En modélisation de données suivant le paradigme ER [14] par exemple, les entités de même type sont représentées par un type d'entité et les liens entre les entités sont représentés par une association entre les types d'entités correspondants. Chaque type d'entité est muni de propriétés décrivant les entités affiliées. À l'instar du modèle ER, nous allons représenter les données de l'ARC par un ensemble de

[5]Dans les modèles conceptuels, la cardinalité d'une relation peut être de type 1-1, 0-N, 1-N, N-N

contextes. Chaque contexte encode des entités ayant des caractéristiques similaires. Ces caractéristiques sont représentées par des attributs formels. Les liens inter-entités sont exprimés par des relations inter-contextes.

Formellement, les données en ARC sont décrites par une collection de contextes, notée $\mathbf{K} = \{\mathcal{K}_i\}$, et un ensemble de relations binaires, noté $\mathbf{R} = \{r_j\}$, représentant les relations d'incidence entre ensembles d'objets de $\mathbf{K}$. Ainsi une relation $r \subseteq O_i \times O_j$ donne lieu à une relation binaire dont les lignes et les colonnes correspondent aux objets de O_i et O_j, respectivement. Formellement, une famille de contextes relationnels (FCR) est définie de la manière suivante :

Définition 39.

Une famille de contextes relationnels (FCR) est une paire $(\mathbf{K}, \mathbf{R})$ *où*

- $\mathbf{K}$ *est un ensemble de contextes pluri-valués* $\mathcal{K}_i = (O_i, A_i, I_i)$,
- $\mathbf{R}$ *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'objets de certains contextes de* $\mathbf{K}$.

Exemple : Le contexte $\mathcal{K}_p$ (Table 4.1) muni de la relation cite (Table 4.2) composent une FCR où $\mathbf{K} = \{\mathcal{K}_p\}$ et $\mathbf{R} = \{\mathsf{cite}\}$.

Par la suite, nous allons considérer que les relations dans $\mathbf{R}$ sont orientées et représentent des fonctions ensemblistes, c'est à dire, $r : O_i \rightarrow 2^{O_j}$. Étant donné une relation r qui va d'un contexte source O_i à un contexte cible O_j, nous allons aussi considérer les fonctions ci-dessous où $\mathbf{O} = \{O_i | \mathcal{K}_i \in \mathbf{K}\}$ et $r : O_i \rightarrow 2^{O_j}$).

- Le domaine d'une relation est le contexte source. $dom : \mathbf{R} \rightarrow \mathbf{O}$, $dom(r) = O_i$,
- Le co-domaine d'une relation est le contexte cible. $cod : \mathbf{R} \rightarrow \mathbf{O}$, $cod(r) = O_j$,
- L'ensemble des relations ratachées à un contexte donné $rel : \mathbf{K} \rightarrow 2^{\mathbf{R}}$, $rel(\mathcal{K}_i) = \{r | dom(r) = O_i\}$.

Exemple : Le domaine et le co-domaine de la relation cite entre les publications est le même contexte, à savoir, le contexte $\mathcal{K}_p$ ($dom(\mathsf{cite}) = cod(\mathsf{cite}) = \mathcal{K}_p$).

Une FCR est comparable à un modèle de données ER ou encore à un schéma d'une base de données relationnelle. La Figure 4.2 illustre le meta-modèle en UML définissant la structure d'une FCR. Ainsi, un contexte binaire est composé d'un en-

semble d'objets formels et d'un ensemble d'attributs formels. Chaque objet formel désigne l'ensemble des attributs formels qu'il possède. Une relation inter-contextes désigne deux contextes formels, à savoir, le contexte source et le contexte cible qui correspondent au domaine et au co-domaine de la relation, respectivement. La relation inter-contextes est aussi composée d'un ensemble de liens qui indiquent les deux objets reliés.

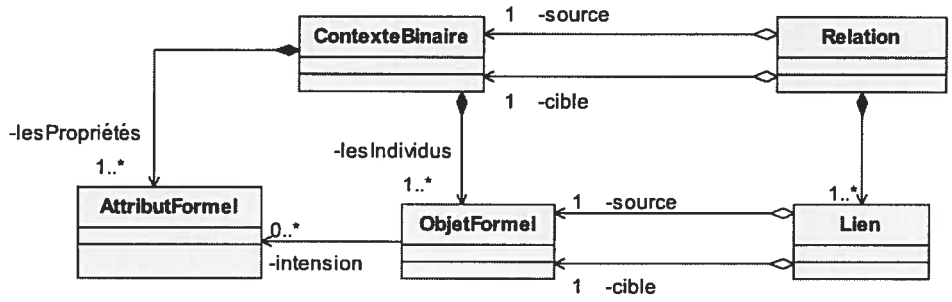


FIG. 4.2 – Le méta-modèle de la FCR.

4.4 FCR comparée aux modèles de données usuels

Comme mentionné ci-dessus, une FCR offre un moyen de représentation similaire à celui fourni par le modèle ER. La Table 4.3 récapitule les correspondances entre la FCR et les différents paradigmes de modélisation de données. Par exemple, les similarités établies avec le modèle de classe de UML [27] révèlent que chaque classe du diagramme correspond à un contexte de la FCR et chaque association correspond à une relation $\{r_j\}$ appartenant à l'ensemble $\mathbf{R}$.

Il ne faut pas confondre cette correspondance avec une autre représentation du modèle UML que nous allons employer dans le cadre de nos expérimentations . Cette dernière, utilisée à des fins de restructuration (voir Chapitre 8.2.2.2), consiste à considérer les classes et les associations comme des objets du méta-modèle UML, c'est-à-dire, des méta-données. De cette manière, les classes et les associations deviennent des objets formels de deux contextes distincts.

FCR	ER	Schéma Relationnel	OO (UML)
contexte formel	entité	table	classe
relation	association	table de jointure[6]	association
objet formel	entité concrète	tuple	instance
attribut formel	attribut	colonne	variable de classe
nom d'objet formel	clé	clé primaire	identifiant d'objet

TAB. 4.3 – Correspondances entre FCR et les modèles de données classiques.

4.5 Traitement des relations en ARC

La question qui se pose en AFC est comment intégrer les relations inter-objets dans le processus d'analyse. Plusieurs approches peuvent être appliquées selon la précision désirée du résultat. La Section 3.1.3 du Chapitre 3 présente quelques unes de ces approches. L'approche *a posteriori* [65] par exemple, délègue le traitement des liens inter-objets à une étape qui interviendrait après la découverte des concepts à partir des descriptions binaires. À ce moment, des relations inter-concepts sont ajoutées en se basant sur les liens entre les objets qu'ils représentent. De plus, d'autres relations inter-concepts peuvent être établies de manière indépendante des liens inter-objets en faisant intervenir la sémantique des concepts par rapport au domaine fourni par les systèmes de structuration de connaissances, ontologies, etc.

Une approche plus intéressante consiste à injecter, dès le départ, les liens inter-objets dans le processus de construction des concepts de façon à ce que les descriptions des concepts trouvés renferment une partie relationnelle inférée à partir du partage des liens. Autrement dit, cette approche permet d'associer des abstractions relationnelles à un concept formel à partir des liens entre les objets de son extension. Ainsi, il faut déterminer pour un concept c d'un contexte donné et une relation r dont le domaine est ce même contexte, l'ensemble des concepts cibles par la relation r appliquée à l'extension de c ($r(Ext(c))$).

Intuitivement, une relation peut être considérée comme un attribut multi-valué et par conséquent prise en charge par un processus de scaling conceptuel. Dans ce qui suit, nous allons montrer comment des échelles de relations sont construites et

comment un nouveau mécanisme de scaling est mis en place permettant la définition de nouveaux attributs formels dans les contextes reliés.

4.5.1 Scaling relationnel

Dans cette section, nous allons introduire la notion de scaling relationnel, son principe, les échelles relationnelles et les différents schémas de codage de ces échelles.

4.5.1.1 Principe

Le scaling conceptuel (voir Chapitre 2, Section 2.1.3.2) est un mécanisme permettant de convertir les contextes pluri-valués en contextes binaires. Ce mécanisme consiste à remplacer les valeurs concrètes des attributs formels pluri-valués par des abstractions. Ainsi, à partir des valeurs concrètes des attributs formels on construit des échelles qui représentent des contextes binaires où les objets formels sont ces valeurs concrètes alors que les attributs formels sont les abstractions significatives correspondantes. Ces abstractions sont généralement définies par un expert du domaine et représentent des groupes de valeurs concrètes. Par exemple, l'attribut pluri-valué age dont les valeurs sont $\{22, 25, 26, 28, 33, 35\}$ peut être transformé en attributs binaires de la forme $age \leq 27$, $age > 27$, etc. (voir Figure 2.2).

Dans le cas des relations inter-contextes, étant donné une relation $r : O_i \rightarrow 2^{O_j}$, chaque objet o_i du contexte O_i est lié à l'ensemble des objets $r(o_i) \subseteq O_j$. Ainsi, une relation inter-contexte peut être assimilée à un attribut multi-valué, d'où l'intuition de lui associer une échelle, dite "*échelle relationnelle*". La Table 4.4 présente la relation cite donnée par la Table 4.2 sous la forme d'un attribut multi-valué. Par exemple, la publication *God*98 cite les deux publications *God*93 et *God*95.

	cite
*Fun*95	$\{Kro94\}$
*God*98	$\{God93, God95\}$
*Til*03*S*	$\{Lin97, Sne98R\}$

TAB. 4.4 – L'attribut multi-valué cite du contexte des publications $\mathcal{K}_p$.

Dans une échelle relationnelle, les objets formels sont des ensembles d'objets de la forme $\{r(o_i)|o_i \in O_i\}$ alors que les attributs formels sont des attributs décrivant les ensembles d'objets de O_j. À noter que l'ensemble des attributs A_j décrit déjà les objets de O_j. Dans notre exemple de publications, $O_j = O_i$ et $A_j = A_i = \{ab, ca, cd, ma\}$. Ainsi, les attributs A_j peuvent être utilisés de manière à associer à l'ensemble $r(o_i) \subseteq O_j$ tous les attributs qui sont partagés par les objets dans $r(o_i)$. Par exemple, les attributs $\{cd, ma\}$ sont utilisés pour décrire $r(Til03S) = \{Lin97, Sne98R\}$. On remarque que les attributs de A_j partagés représentent l'intension $r(o_i)'$ du concept le plus spécifique dans le treillis $\mathcal{L}_j$ qui contient $r(o_i)$. Dans notre exemple des publications, c'est le concept $c_{\#0}$ du treillis $\mathcal{L}_p$ donné par la Figure 4.1. Nous pensons donc que les concepts sur le contexte $\mathcal{K}_j$ peuvent être utilisés pour décrire les valeurs de la relation r car, d'une part, un concept est une abstraction meilleure qu'un ensemble d'attributs, et d'autre part, dans le cas particulier du contexte des publications les attributs formels du co-domaine de la relation cite sont déjà utilisés pour décrire les objets formels du domaine de cette même relation.

Utiliser des concepts comme des attributs formels de l'échelle d'une relation r permet aux objets du domaine de la relation d'être directement liés aux objets présents dans l'extension de ces concepts au lieu d'être lié à des objets dont la description correspond aux attributs du contexte associé. Par exemple, dans la construction de l'échelle de la relation cite, au lieu d'avoir des publications (objets de l'échelle) qui citent d'autres publications inconnues mais dont la description correspond aux activités de génie logiciel énoncées (attributs de l'échelle), il est plus intéressant d'avoir des publications qui sont liées aux concepts (attributs de l'échelle) qui décrivent les publications qu'ils citent. Par conséquent, le treillis $\mathcal{L}_j$ du co-domaine est d'une importance cruciale dans le scaling d'une relation r car il représente la source des abstractions sur l'ensemble des objets O_j.

Exemple : La Table 4.5 montre une partie de l'échelle relationnelle de la relation cite. Les objets formels de l'échelle sont les sous ensemble $r(o_i)$ avec o_i une publication du contexte $\mathcal{K}_p$. Les attributs formels sont des concepts du treillis $\mathcal{L}_p$

de la Figure 4.1. Une occurrence $r(o_i) \times c_j$ de la relation binaire est positionnée à 1 quand les objets dans $r(o_i)$ font partie de l'extension du concept c_j. Nous allons voir dans la section suivante les différentes manières de construction des échelles relationnelles.

	$c_{\#0}$	$c_{\#1}$	$c_{\#2}$	$c_{\#3}$	$c_{\#4}$	$c_{\#5}$	$c_{\#6}$
$\{God93(2)\}$	1	0	1	0	1	0	0
$\{God93(2), God95(3)\}$	1	0	1	0	1	0	0
$\{Kro94(7)\}$	1	0	1	0	1	0	0
$\{Lin97(11), Sne98R(16)\}$	1	0	1	0	1	0	0

TAB. 4.5 – Échelle de la relation cite utilisant le treillis du contexte des publications $\mathcal{L}_p$.

L'utilisation des échelles relationnelles pour le codage des liens inter-objets dans les contextes définit une nouvelle forme de scaling, dit "*scaling relationnel*". Rappelons que nous nous intéressons à la caractérisation de l'ensemble des valeurs de la relation r pour un concept c du treillis du contexte. Cet ensemble est le co-domaine de r restreint aux seuls objets en relation avec des objets de l'extension de c. Il est défini comme suit :

$$r(c) = \{o \in O_j \mid \exists\, \bar{o} \in Ext(c),\ o \in r(\bar{o})\}.$$

Par exemple, $\mathsf{cite}(c_{\#3}) = \{Lin97, Sne98R\}$. Les échelles relationnelles des différentes relations sont exploitées par le mécanisme de scaling relationnel qui permet de traduire l'ensemble des relations $rel(\mathcal{K}_i)$ d'un contexte $\mathcal{K}_i$ en attributs binaires. Ainsi, étant donné une relation r et soient i et j tel que $r : O_i \rightarrow 2^{O_j}$, nous allons nous servir des attributs de l'échelle relationnelle de r qui sont des concepts c du treillis $\mathcal{L}_j$ pour créer des attributs binaires de la forme "$r : c$" dans le contexte $\mathcal{K}_i$. Ces nouveaux attributs sont appelés '*attributs relationnels*'. Toutefois, il y a différentes façons d'affecter les attributs relationnels aux objets formels du contexte $\mathcal{K}_i$ appelées '*schémas de codage*' ou encore '*mode de codage*'. Dans ce qui suit, nous allons présenter deux schémas de codage, dits 'étroit' et 'large'.

4.5.1.2 Schémas de codage de l'échelle relationnelle

Le scaling d'une relation r dans un contexte $\mathcal{K}_i = (O_i, A_i, I_i)$, avec $cod(r) = \mathcal{K}_j$, produit un enrichissement de A_i et de I_i. Toutefois il y a plusieurs manières de coder l'échelle relationnelle, dépendamment de comment on veut comparer les sous-ensembles $r(o)$ avec les extensions $Ext(c)$ des concepts du treillis $\mathcal{L}_j$.

4.5.1.2.1 Codage étroit : Dans ce mode de codage, un objet donné o de O_i reçoit l'attribut relationnel $r : c$ si est seulement si l'image de o par la relation r est incluse dans l'extension du concept c. Ainsi, étant donné un contexte $\mathcal{K}_i = (O_i, A_i, I_i)$, une relation r, tel que $dom(r) = \mathcal{K}_i$ et le treillis $\mathcal{L}_j$ associé au contexte $\mathcal{K}_j = cod(r)$, l'opérateur de codage étroit, noté $sc_{\subseteq}^{(r,\mathcal{L}_j)}$, produit en sortie le contexte $\mathcal{K}_i$ enrichi avec l'ensemble $\{r : c | c \in \mathcal{L}_j\}$ des attributs relationnels vérifiant la conditions d'inclusion ci-dessous. La nouvelle version du contexte $\mathcal{K}_i$ comporte un plus grand ensemble d'attributs formels, noté $A_i^{(r,\mathcal{L}_j)}$ et une relation binaire augmentée par les paires $(o, r : c)$, notée $I_i^{(r,\mathcal{L}_j)}$. Formellement, l'opérateur de codage étroit est défini de la manière suivante :

Définition 40. *Étant donné une relation $r \in rel(\mathcal{K}_i)$ et un treillis $\mathcal{L}_j$ associé au contexte $\mathcal{K}_j = cod(r)$, l'opérateur de codage étroit $sc_{\subseteq}^{(r,\mathcal{L}_j)}$ est défini de la façon suivante :*

$$sc_{\subseteq}^{(r,\mathcal{L}_j)} : \mathbf{K} \rightarrow \mathbf{K}$$

$$sc_{\subseteq}^{(r,\mathcal{L}_j)}(\mathcal{K}_i) = (O_i^{(r,\mathcal{L}_j)}, A_i^{(r,\mathcal{L}_j)}, I_i^{(r,\mathcal{L}_j)}), \text{ où :}$$

- $O_i^{(r,\mathcal{L}_j)} = O_i$,
- $A_i^{(r,\mathcal{L}_j)} = A_i \cup \{r : c | c \in \mathcal{L}_j\}$,
- $I_i^{(r,\mathcal{L}_j)} = I_i \cup \{(o, r : c) | o \in O_i, c \in \mathcal{L}_j, r(o) \subseteq Ext(c)\})$.

Exemple : Considérons le scaling du contexte des publications de la Table 4.1 le long de la relation circulaire cite illustré par la Table 4.2 en utilisant le treillis initial de la Figure 4.1 obtenu à partir des attributs binaires du contexte des publications. La Table 4.6 présente le résultat du scaling de la relation cite en utilisant le treillis du contexte des publications de la Figure 4.1 et un mode de codage étroit.

	(1) Fun95	(2) God93	(3) God95	(4) God98	(5) Huc99	(6) Huc02	(7) Kro94	(8) Kui00	(9) Leb99	(10) Lin95	(11) Lin97	(12) Sah97	(13) Sif97	(14) Sne96	(15) Sne98C	(16) Sne98R	(17) Sne99	(18) Sne00S	(19) Sne00U	(20) Str99	(21) Til03S	(22) Til03T	(23) Ton99	(24) Tone01	(25) Van98
cite:0	1	0	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
cite:1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
cite:2	1	0	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
cite:3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
cite:4	1	0	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
cite:5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
cite:6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TAB. 4.6 – La transposée de l'extension relationnelle du contexte des publications obtenue par le scaling de la relation cite avec un mode de codage étroit.

4.5.1.2.2 Codage large : Ce mode de codage relaxe la contrainte d'inclusion imposée par le mode de codage étroit entre l'image d'un objet o de O_i par la relation r et l'extension du concept c qui est désigné par l'attribut relationnel $r : c$. En effet, o reçoit l'attribut relationnel $r : c$ si et seulement si l'intersection entre $r(o)$ et $Ext(c)$ est non vide. L'opérateur de codage large, noté $sc_{\cap}^{(r,\mathcal{L}_j)}$, est défini de manière similaire au codage étroit avec une légère différence dans le calcul du contexte de l'échelle.

Étant donné un contexte $\mathcal{K}_i = (O_i, A_i, I_i)$, une relation r, tel que $dom(r) = \mathcal{K}_i$ et le treillis $\mathcal{L}_j$ associé au contexte $\mathcal{K}_j = cod(r)$, l'opérateur de codage large produit en sortie le contexte $\mathcal{K}_i$ enrichi avec l'ensemble des attributs relationnels $\{r : c | c \in \mathcal{L}_j\}$ vérifiant la condition d'intersection ci-dessous. À l'instar du codage étroit,la nouvelle version du contexte $\mathcal{K}_i$ comporte un plus grand ensemble d'attributs formels et une relation binaire augmentée. Formellement, l'opérateur de codage large est défini de la façon suivante :

Définition 41. *Étant donné une relation $r \in rel(\mathcal{K}_i)$ et un treillis $\mathcal{L}_j$ associé au contexte $\mathcal{K}_j = cod(r)$, l'opérateur de codage large $sc_{\cap}^{(r,\mathcal{L}_j)}$ est défini de la façon suivante :*

$$sc_{\cap}^{(r,\mathcal{L}_j)} : \mathbf{K} \rightarrow \mathbf{K}$$

$$sc_{\cap}^{(r,\mathcal{L}_j)}(\mathcal{K}_i) = (O_i^{(r,\mathcal{L}_j)}, A_i^{(r,\mathcal{L}_j)}, I_i^{(r,\mathcal{L}_j)}),\ où :$$

- $O_i^{(r,\mathcal{L}_j)} = O_i$,
- $A_i^{(r,\mathcal{L}_j)} = A_i \cup \{r : c | c \in \mathcal{L}_j\}$,
- $I_i^{(r,\mathcal{L}_j)} = I_i \cup \{(o, r : c) | o \in O_i, c \in \mathcal{L}_j, r(o) \cap Ext(c) \neq \emptyset\})$.

Le scaling de la relation cite en utilisant le treillis du contexte des publications de la Figure 4.1 et en mode de codage large est identique à celui illustré par la Table 4.6. À noter que dans le cas où aucune précision n'est donnée sur le mode de codage utilisé, un codage étroit est considéré. De plus, on ne compte pas mélanger les deux modes de codage d'où la notation unique de l'attribut relationnel, mais si c'était le cas, on pourrait les noter $\exists\ r : c$ et $\forall\ r : c$ pour désigner un attribut relationnel obtenu par le mode de code large et étroit, respectivement. On peut aussi, à l'instar des logiques de descriptions, avoir des codages avec des restrictions numériques. Ainsi, un objet o ayant des liens de type r avec au moins n objets d'un concept c donne lieu à un attribut relationnel du genre $r : c[n]$.

4.5.2 Le contexte étendu

Un scaling relationnel complet d'un contexte $\mathcal{K}_i$ est obtenu par le scaling de toutes les relations de l'ensemble $rel(\mathcal{K}_i)$. Ce scaling produit un contexte plus volumineux, appelé "contexte *étendu*".

Étant donné un contexte $\mathcal{K}_i$, tel que $rel(\mathcal{K}_i) = \{r_l\}_{l=1..p_i}$, et l'ensemble de treillis $\mathcal{L}_{j_l}$ associés aux contextes composant les différents co-domaines de r_l pour tout l dans $[1, p_i]$, le contexte étendu $\mathcal{K}_i^{rel}$ obtenu à partir de $\mathcal{K}_i$ par un scaling relationnel complet, c'est à dire tous les paires $(r_l, \mathcal{L}_{j_l})$, est dénoté comme suit :

$$\mathcal{K}_i^{rel} = (sc_{\subseteq}^{(r_1,\mathcal{L}_{j_1})}(sc_{\subseteq}^{(r_2,\mathcal{L}_{j_2})}(\ldots sc_{\subseteq}^{(r_{k_i},\mathcal{L}_{j_{p_i}})}(\mathcal{K}_i^0)))).$$

$\mathcal{K}_i^0$ représente le contexte obtenu à partir du contexte initial $\mathcal{K}_i$ par le scaling conceptuel des attributs pluri-valués.

Exemple : La Table 4.7 illustre le contexte des publications étendu obtenu par le scaling de la relation cite en utilisant le treillis initial du même contexte.

	(1) Fun95	(2) God93	(3) God95	(4) God98	(5) Huc99	(6) Huc02	(7) Kro94	(8) Kui00	(9) Leb99	(10) Lin95	(11) Lin97	(12) Sah97	(13) Sif97	(14) Sne96	(15) Sne98C	(16) Sne98R	(17) Sne99	(18) Sne00S	(19) Sne00U	(20) Str99	(21) Til03S	(22) Til03T	(23) Ton99	(24) Tone01	(25) Van98
ab	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
ca	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
cd	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
ma	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1
cite:0	1	0	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
cite:1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
cite:2	1	0	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
cite:3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
cite:4	1	0	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
cite:5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
cite:6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TAB. 4.7 – La transposée du contexte formel étendu des publications obtenu par un mode de codage étroit de la relation cite.

Une fois que tous les contextes de la FCR ont été étendus, il y a des chances que l'extension relationnelle courante d'un contexte $\mathcal{K}_j$ ne soit plus conforme à l'extension précédente $sc_{\subseteq}^{(r,\mathcal{L}_j)}$. En effet, le mécanisme de scaling relationnel, tel que décrit ci-dessus, s'appuie fortement sur la disponibilité des treillis de tous les contextes connectés au contexte $\mathcal{K}_i$ qui fait l'objet d'une extension. Toutefois, quand $\mathcal{K}_i$ est étendu, le treillis correspondant est probablement plus volumineux que celui qui correspond au contexte initial (avant extension). Cela signifie qu'avec une FCR munie d'un réseau complexe de relations, tout scaling d'un contexte donné $\mathcal{K}_m$ le long d'une relation r_m, avec $cod(r) = O_i$, précédant l'extension de $\mathcal{K}_i$ n'est plus à jour et par conséquent doit être ré-actualisé. Éviter des cas pareils revient à imposer une structure sans circuits du multi-graphe orienté et étiquetté[7] à la FCR. En d'autres termes, il faut qu'il ait un ordre total sur les tâches de scaling des contextes permettant d'éviter le re-scaling. Cependant, ce serait une contrainte trop forte car beaucoup de cas intéressants resteraient en dehors de la portée de l'approche (par exemple, la FCR composée du contexte des publications et de la relation de citation

[7]Les nœuds et les arcs sont les contextes et les relations de la FCR, respectivement.

entre publications).

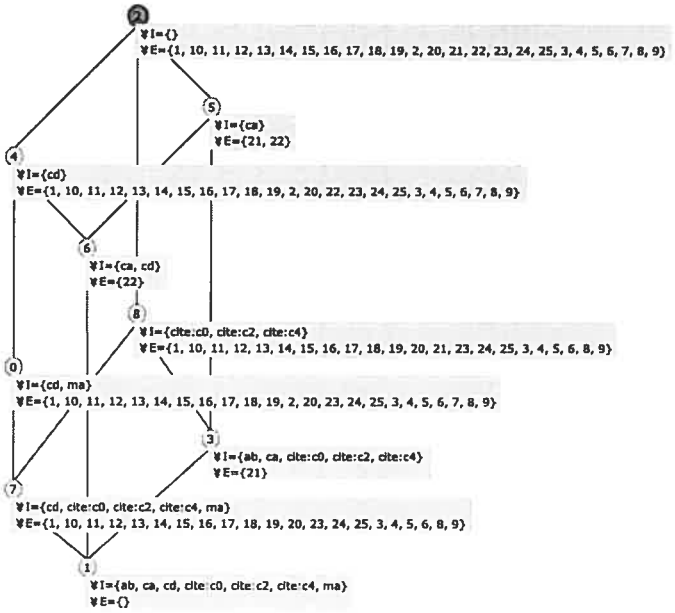


FIG. 4.3 – Le treillis relationnel $\mathcal{L}_p^1$ du contexte illustré par la Table 4.7.

Dans la section suivante nous présenterons une méthode générale de construction de treillis d'une FCR fonctionnant même en présence de circuits.

4.6 Construction des treillis de la FCR

4.6.1 Principe de la méthode itérative Multi-FCA

La méthode proposée, appelée "MULTI-FCA", permet de construire les treillis des contextes de la FCR en deux principales phases. Dans la première phase (initialisation), pour tout contexte $\mathcal{K}_i$ pluri-valué de la FCR, un contexte binaire $\mathcal{K}_i^0$ est produit par scaling conceptuel. Le treillis correspondant est aussitôt construit. La deuxième phase consiste à calculer les contextes étendus à partir des différentes relations de la FCR et à mettre à jour les treillis respectifs. Ainsi, la méthode applique

une suite d'itérations alternant entre le scaling relationnel des différentes relations et la re-construction des treillis des contextes qui été étendus suivant ces relations. Le processus se termine dès qu'aucun concept n'est créé durant la re-construction des treillis de la FCR. En d'autres termes, les treillis de la nouvelle itération sont isomorphes à ceux de l'itération précédente. À ce moment, on dit qu'un point fixe de l'opérateur de scaling $sc_{\subseteq}^{(r,\mathcal{L}_j)}$ est atteint pour tout r et $\mathcal{L}_j$ de $(\mathbf{K}, \mathbf{R})$.

À chaque itération p du processus itératif, le contenu du contexte $\mathcal{K}_i$ est dénoté par $\mathcal{K}_i^p$. Formellement, la méthode peut être décrite de la manière suivante :

Étape (0) : $\mathcal{L}_i^0$ est construit à partir de $\mathcal{K}_i^0$, le contexte obtenu en transformant tous les attributs pluri-valués du contexte initial $\mathcal{K}_i$ à l'exception des relations (r),

Étape $(p+1)$: $\mathcal{L}_i^p$ est utilisé comme échelle pour la relation r afin de produire le contexte $\mathcal{K}_i^{p+1}$ et ensuite le treillis correspondant $\mathcal{L}_i^{p+1}$.

Critère d'arrêt : la procédure s'arrête quand $\mathcal{L}_i^{n+1}$ est isomorphe à $\mathcal{L}_i^n$ pour tout i. La limite de cette suite est dénotée par $\mathcal{L}_i^\infty$.

À chaque itération, les contextes de la FCR subissent une extension relationnelle complète. Formellement, on définit l'opérateur d'extension complète de la manière suivante :

Définition 42. *Étant donné $\mathcal{K}_i^p$, le i^{eme} contexte de la FCR à l'étape p du processus itératif, l'opérateur $.^{rel_p}$ retourne le scaling relationnel complet du contexte $\mathcal{K}_i^p$:*

$$\mathcal{K}_i^{p+1} = (\mathcal{K}_i^p)^{rel_p},$$
$$\mathcal{K}_i^0 = (O_i, A_i^0, I_i^0).$$

Le processus itératif produit pour chaque contexte une série de versions. La version du contexte à l'étape $p+1$ est obtenue par un scaling relationnel complet qui consiste, rappelons le, à étendre la version de l'étape p avec les échelles à l'étape p aussi des différentes relations dans $rel(\mathcal{K}_i)$. Le pseudo-code de la méthode est présenté dans le Chapitre 5, Section 5.1.2.

Exemple : La Table 4.7 et la Table 4.8 illustrent le contexte des publications $\mathcal{K}_p$ au niveau de la première et la seconde étape du processus itératif, respectivement.

	(1) Fun95	(2) God93	(3) God95	(4) God98	(5) Huc99	(6) Huc02	(7) Kro94	(8) Kui00	(9) Leb99	(10) Lin95	(11) Lin97	(12) Sah97	(13) Sif97	(14) Sne96	(15) Sne98C	(16) Sne98R	(17) Sne99	(18) Sne00S	(19) Sne00U	(20) Str99	(21) Til03S	(22) Til03T	(23) Ton99	(24) Tone01	(25) Van98
ab	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
ca	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
cd	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
ma	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1
cite:0	1	0	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
cite:1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
cite:2	1	0	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
cite:3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
cite:4	1	0	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
cite:5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
cite:6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
cite:7	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	1
cite:8	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	1

TAB. 4.8 – La transposée du contexte formel des publications étendu obtenu par un mode de codage étroit de la relation cite après la seconde itération ($\mathcal{K}_p^2$).

Le contexte $\mathcal{K}_i^\infty$ représente le point fixe d'une série de versions du contexte $\mathcal{K}_i$. Dans notre exemple, $\mathcal{K}_p^\infty = \mathcal{K}_p^2$. Cette série est de nature non décroissante du point de vue de la taille de l'ensemble des attributs A_i. Formellement, on a la propriété suivante :

Propriété 18. $\forall\ \mathcal{K}_i \in \mathbf{K}$, *si* $\mathcal{K}_i^p = (O_i, A_i^p, I_i^p)$ *et* $\mathcal{K}_i^{p+1} = (O_i, A_i^{p+1}, I_i^{p+1})$, *alors* $A_i^p \subseteq A_i^{p+1}$ *et* $I_i^p \subseteq I_i^{p+1}$.

En effet, le scaling qui est à l'origine du versionnement peut seulement ajouter des attributs formels aux contextes ou les laisser inchangés. De plus, pour exprimer la dynamique de l'ensemble de contextes $\mathbf{K}$, nous introduisons le vecteur de contextes $\mathbb{K}$ et un opérateur de composition, dénoté $.^{rel_p^*}$, agissant sur ce vecteur. Cet opérateur consiste à appliquer l'opérateur $.^{rel_p}$ à tous les contextes de ce vecteur pour produire une série $\mathbb{K}^n$ de versions de contextes. La série $\mathbb{K}^n$ de versions de contextes est définie de la manière suivante :

$$\mathbb{K}^0 = \mathbf{K},$$
$$\mathbb{K}^{p+1} = (\mathbb{K}^p)^{rel_p^*},\ avec\ p \geq 0.$$

Il faut noter que la série $\mathbb{K}^n$ résultante est bornée car les séries composantes $\mathcal{K}_i^n$ sont bornées ($\mathcal{K}_i^\infty$) (voir Section 4.6.5). La série est aussi non décroissante car elle possède une limite, dénotée $\mathbb{K}^\infty$. Cette limite est le point où les opérateurs de scaling relationnel ne produisent plus de nouveaux concepts dans aucun contexte.

Exemple : La Figure 4.4 et la Figure 4.5 représentent le treillis final $\mathcal{L}_p^\infty$ obtenu à partir du contexte des publications avec un scaling relationnel utilisant un mode de codage étroit et large, respectivement.

Dans le chapitre suivant, nous allons décrire un algorithme général permettant d'étendre la portée du principe ci-dessus à une FCR entière.

4.6.2 Étiquetage relationnel réduit

Le mécanisme de scaling relationnel produit des attributs relationnels qui serviront au calcul de nouvelles similarités entre objets formels. Ainsi, les concepts qui représentent ces similarités pointent vers d'autres concepts de la FCR. Un concept relationnel est défini de la façon suivante : étant donné un contexte $\mathcal{K} = (O, A, I)$, $\mathcal{K} \in \mathbf{K}$ et le treillis correspondant $\mathcal{L}_\mathcal{K} = \langle \mathcal{C}_\mathcal{K}, \leq_\mathcal{K} \rangle$, un concept relationnel c de $\mathcal{C}_\mathcal{K}$ est une paire (X, Y) où $X \subseteq O$ et $Y = a_1 a_2 \ldots a_n r_1 r_2 \ldots r_m$, tel que $a_i \in A$ et r_i est un attribut formel binaire ayant la forme $r : c$ qui signifie une liaison de type $r \in \mathbf{R}$ avec un concept $c \in \mathcal{L}_j$. Le sous ensemble d'attributs $a_1 a_2 \ldots a_n$ est noté $Int_l(c)$ ('l' pour local) alors que $a_n r_1 r_2 \ldots r_m$ est noté $Int_r(c)$ ('r' pour relationnel).

Exemple : Le concept $c_{\#3} = (\{21\}, \{ab, ca, \text{cite:c0}, \text{cite:c2}, \text{cite:c4}, \text{cite:c7}, \text{cite:c8}\})$ (voir Figure 4.4) est un concept relationnel où $Int_l(c) = \{ab, ca\}$ et $Int_l(c) = \{\text{cite:c0}, \text{cite:c2}, \text{cite:c4}, \text{cite:c7}, \text{cite:c8}\}$. Comme la relation cite est entre publications, tous les concepts désignés appartiennent au même treillis $\mathcal{L}_p$.

Par ailleurs, les opérateurs de codage étroit et large du mécanisme de scaling produisent des interprétations redondantes des attributs relationnels. En effet la présence d'une relation avec un concept donné entraine une relation avec tous ses

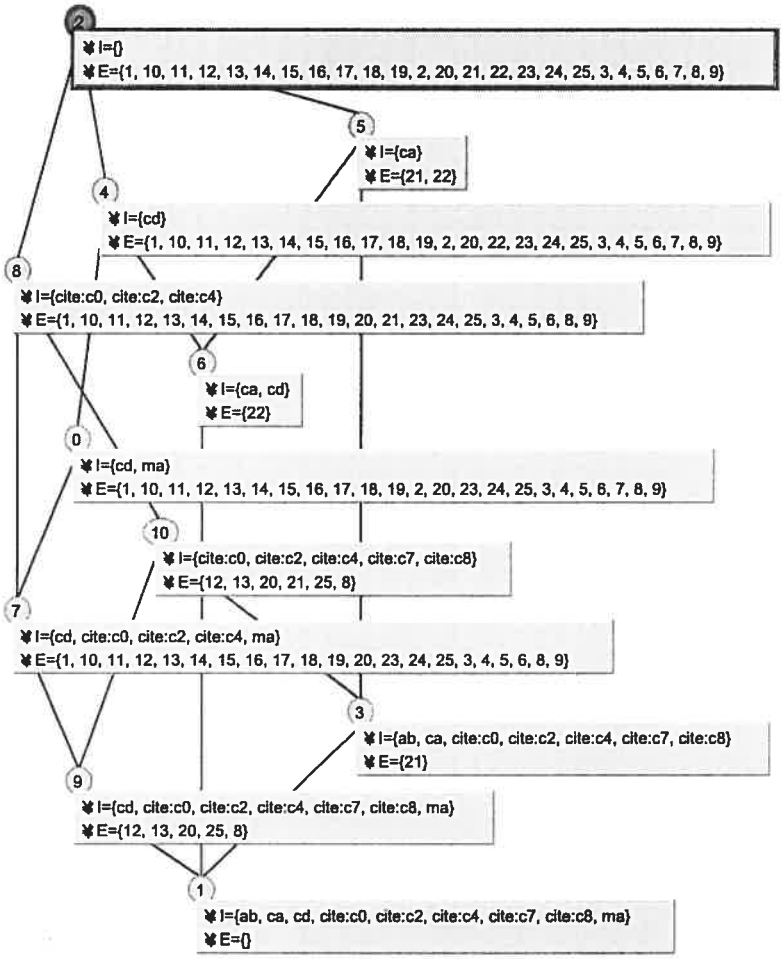


FIG. 4.4 – Le treillis relationnel final $\mathcal{L}_p^\infty$ obtenu à partir du contexte des publications avec un scaling relationnel étroit.

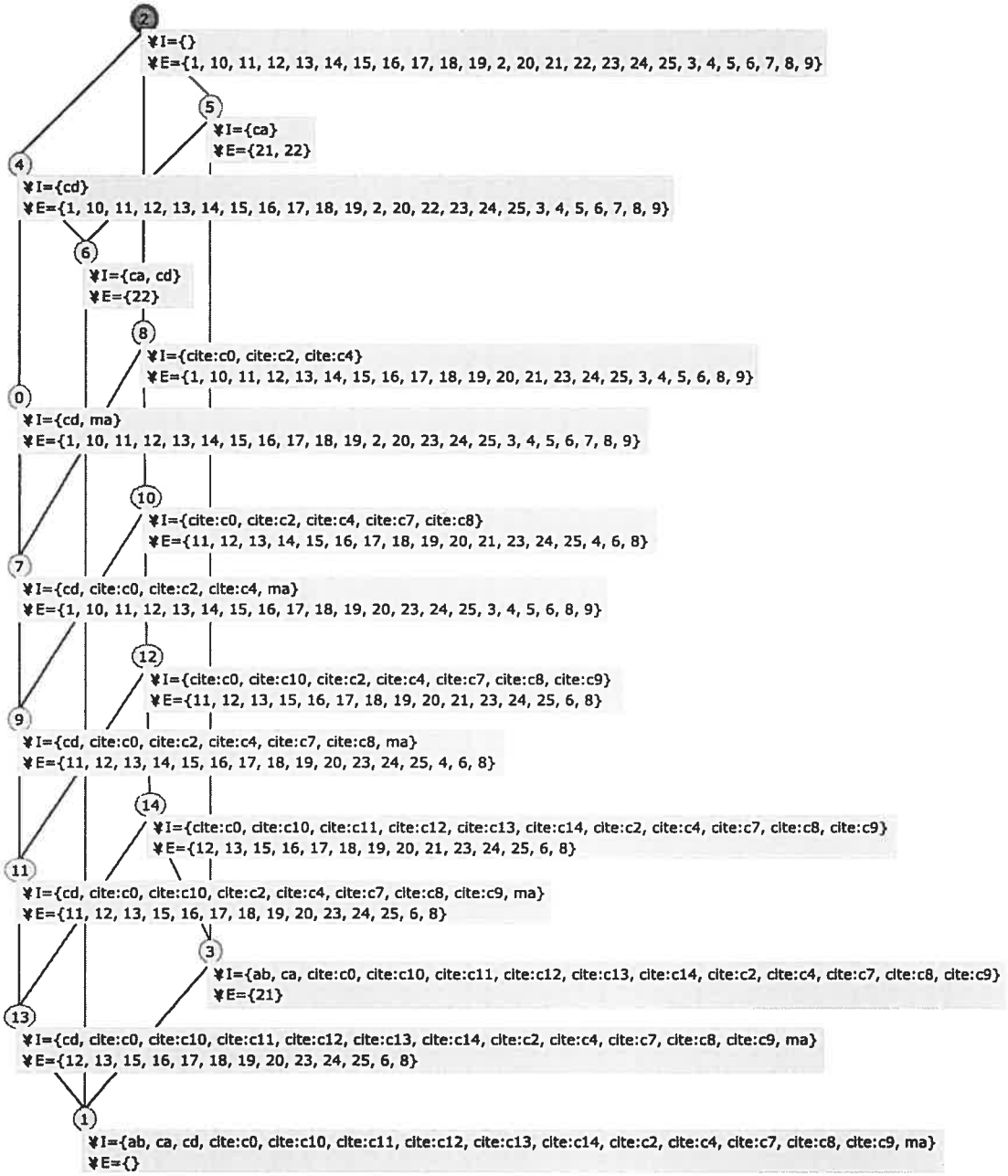


FIG. 4.5 – Le treillis relationnel final $\mathcal{L}_p^\infty$ obtenu à partir du contexte des publications avec un scaling relationnel large.

successeurs dans le treillis.

Propriété 19. *Étant donné un concept c du treillis $\mathcal{L}_i$ associé au contexte $\mathcal{K}_i$ et pour toute relation $r \in rel(\mathcal{K}_i)$ avec $cod(r) = \mathcal{K}_j$. L'étiquetage relationnel réduit de c, noté $Int_r^*(c)$, est obtenu de la manière suivante :*

$$Int_r^*(c) = Int_r(c) \setminus \{r : \bar{c} \mid \exists \hat{c} \in \mathcal{L}_j, \hat{c} \leq \bar{c}, r : \hat{c} \in Int_r(c).\}$$

.

Démonstration. Supposons que $\{r : \hat{c}, r : \bar{c}\} \subseteq Int_r(c)$ avec $\hat{c} \leq \bar{c}$. Puisque $\{r : \hat{c}\} \subset Int_r(c)$ alors $\forall o \in Ext(c)$ on a $r(o) \subseteq Ext(\hat{c})$. Comme $\forall c_1 \forall c_2, c_1 \leq_{\mathcal{K}} c_2, Ext(c_1) \subseteq Ext(c_2)$ alors $r(o) \subseteq Ext(\bar{c})$. Par conséquent, $\{r : \bar{c}\} \subset Int_r(c)$. Ainsi, il suffit d'avoir l'attribut relationnel $r : \hat{c}$ dans $Int_r^*(c)$ pour déduire la présence du même type d'attributs relationnels désignant son successeur $\bar{c}$. □

On peut simplifier l'ensemble $Int_r(c)$ dans le cas de la présence de plusieurs attributs relationnels de même type dont les concepts désignés sont comparables. La simplification consiste à remplacer tous ces attributs par les attributs relationnels dont les concepts spécifiés sont minimaux. À noter que l'élimination des attributs relationnels dits 'redondants' produit une structure avec une représentation plus compacte et lisible mais qui ne préserve pas la structure de treillis.

La Figure 4.6 montre le treillis final du contexte des publications $\mathcal{L}_p^\infty$ avec un étiquetage simplifié des ensembles $Int_l(c)$ et une réduction des ensembles $Int_r(c)$.

4.6.3 Treillis et navigation relationnelle

Un treillis de concepts représente un moyen efficace de navigation et d'interrogation dans le contexte qui lui correspond. Le treillis relationnel vient ajouter une nouvelle dimension à cette navigabilité en introduisant des relations inter-concepts induites à partir des liens inter-objets.

Par exemple, dans le cas du contexte des publications, le treillis final $\mathcal{L}_p^\infty$ (voir Figure 4.7) fournit une vue du patron de citation dans le contexte des publications.

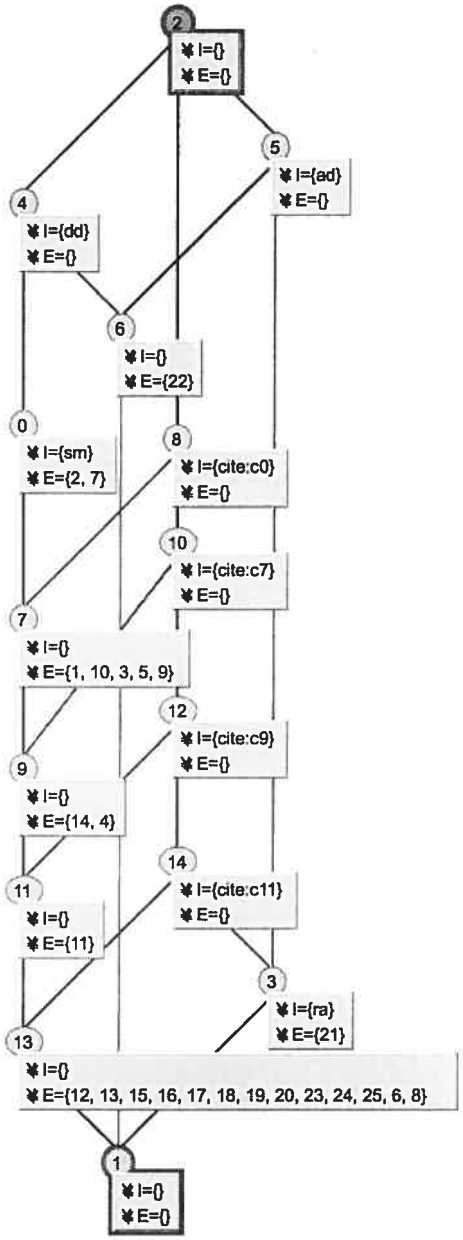


FIG. 4.6 – Étiquetage relationnel réduit du treillis $\mathcal{L}_p^\infty$ de la Figure 4.5.

En effet, en observant par exemple les deux concepts inter-reliés $c_{\#3}$ et $c_{\#7}$, on pourrait avoir une bonne indication des publications qui traitent de l'analyse des besoins et qui font référence aux étapes suivantes dans le cycle de développement du logiciel telles que la conception détaillée et la maintenance. Le concept-objet $c_{\#3}$ représente la publication de T. Tilley sur l'analyse des besoins et conception architecturale (objet formel 21 ou encore $Til03S$ de la Table 4.1). Selon la Table 4.2, cette publication référence seulement la publication de G. Snelting sur la réingénierie des hiérarchies de classes (objet formel 11 ou encore $Sne98R$ de la Table 4.1) et celle de C. Lindig avec G. Snelting sur l'étude de la structure modulaire du code légataire (objet formel 16 ou encore $Lin97$ de la Table 4.1). En considérant maintenant la relation cite, le cercle de la citation est élargi à d'autres publications qui sont représentées par le concept $c_{\#7}$. En effet, le concept $c_{\#7}$ représente le sous ensemble maximal de publications qui traitent, à la fois, la conception détaillée et la maintenance et qui partagent un certains nombre de citations.

D'autres connaissances peuvent être extraites en combinant la navigation suivant les relations dans un treillis relationnels et la connaissance du domaine à partir duquel le contexte est produit. Dans notre exemple des deux concepts inter-reliés $c_{\#3}$ et $c_{\#7}$, il pourrait s'agir de certaines pratiques en analyse de besoins (concept $c_{\#3}$) qui affectent les activités de conception et de maintenance du logiciel($c_{\#7}$). Un autre patron relationnel induit par les liens inter-publications dans le treillis de la Figure 4.7 est le fait que toute publication qui traite de l'analyse de besoins doit impérativement évoquer la conception détaillée et la maintenance du logiciel.

4.6.4 Représentation graphique du treillis relationnel

La représentation graphique d'une grande famille de treillis relationnels nécessite la mise au point d'algorithmes de dessin appropriés. En effet, de plus la difficulté de faire un minimum de croisement d'arêtes lors de la représentation graphique la relation de précédence dans un treillis, il y a la complexité de représenter les relations entre les concepts des treillis de la famille. Pour dessiner graphiquement des

treillis relationnels, on utilise deux types d'arêtes entre les nœuds qui représentent les concepts, à savoir les arcs relationnels (segments courbés) qui décrivent les relations inter-concepts et ceux classiques (segments droits) qui décrivent la relation de couverture entre concepts (relation d'ordre). Les arcs relationnels sont orientés et étiquetés par un nom de relation pour indiquer que tous les objets dans l'extension du concept de départ possède un lien de type nom de la relation vers chaque objet de l'extension du concept d'arrivée. La Figure 4.7 présente le treillis des publications de la Figure 4.4 sous une forme graphique plus lisible.

4.6.5 Élements de calculabilité et complexité

Afin de clarifier la convergence du processus itératif considérons une FCR composée d'un seul contexte $\mathcal{K} = (O, A, I)$ et d'une seule relation r avec $dom(r) = cod(r) = \mathcal{K}$. Le treillis initial $\mathcal{L}^0$ représente la fermeture par intersection de tous les sous-ensembles d'objets $E_0 = \{a' \mid a \in A\}$, dénotée $C^{o,0}$, tel que, $C^{o,0} = E_0^{\cap}$. Durant le passage de l'étape p à l'étape $p+1$ du processus itératif, le mécanisme de scaling étroit ou large de la relation r applique la fonction inverse r^{-1} à tous les sous-ensembles d'objets $C^{o,p}$ calculés depuis le début du processus itératif jusqu'à l'itération p. La formation de concepts à l'étape $p+1$ consiste à déterminer de nouveau la fermeture de tous les sous-ensembles d'objets par intersection. Ainsi :

$$C^{o,p+1} = (C^{o,p} \cup r^{-1}(C^{o,p}))^{\cap}$$

Quand l'ensemble $C^{o,p}$ se stabilise, un point fixe est atteint et est noté $C^{o,\infty}$. D'autre part, en examinant la méthode de construction (Section 4.6.1), on peut considérer le processus itératif comme une série d'approximations de la solution finale en termes de contextes formels où chaque contexte $\mathcal{K}_i^p$, à l'exception du premier $\mathcal{K}_i^0$, est une extension du précédent $\mathcal{K}_i^{p-1}$. Comme l'ensemble des objets O_i de chaque contexte est invariable, seul l'ensemble des attributs A_i augmente en taille à travers les itérations par l'ajout des attributs relationnels (voir la propriété 18). Par conséquent, entre deux étapes, la taille du treillis d'un contexte donné est

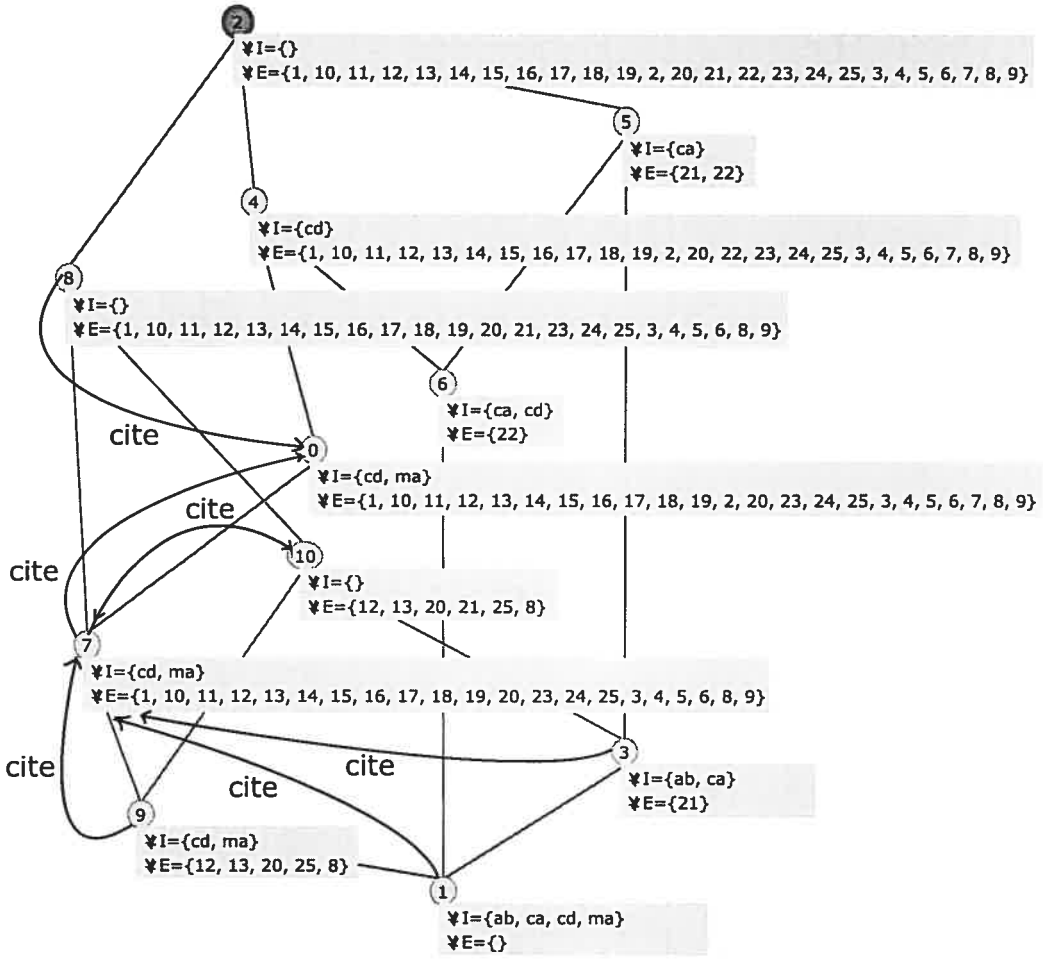


FIG. 4.7 – Représentation graphique du treillis relationnel $\mathcal{L}_p^{\infty}$ de la Figure 4.4.

non décroissante. Toutefois, rappelons que le nombre de concepts d'un treillis est borné par la taille de l'ensemble des parties de l'ensemble des objets du contexte, $|\mathcal{C}_i^p| \leq |2^{O_i}|$, qui lui reste constant. Pour cela, la méthode MULTI-FCA se termine toujours.

Dans la section suivante, nous allons expliquer la convergence du processus itératif par le biais d'un modèle d'objets qui montre le partage d'attributs qui émerge à travers les différentes itérations.

4.7 Rapports entre les résultats et les données de départ

Nous allons donner, dans les paragraphes suivants, une caractérisation orientée données qui clarifie la nature exacte des treillis finaux obtenus par l'ARC. Nous commençons par un ensemble de définitions préliminaires supportant la généralisation du partage d'attributs entre objets qui représente le principe de base de la formation des concepts en AFC ; ensuite, nous donnerons la propriété de formation de ces concepts.

4.7.1 Graphes d'objets

Intuitivement et indépendamment de l'étape de processus itératif, deux objets appartenant à un même contexte $\mathcal{K}_i$ ne peuvent appartenir à un même concept du treillis que s'ils partagent au moins un attribut, que ce soit un attribut initial ou un attribut en provenance d'une échelle relationnelle. Afin de formaliser la notion de la structure partagée qui peut impliquer des attributs ou des liens, nous introduisons la notion de graphes d'objets. Ces graphes sont des structures couvrant tous les objets de la FCR qui peuvent être atteints en partant d'un objet particulier et en suivant ses liens. Nous introduisons d'abord la fonction *ctx* permettant de retourner pour un objet o donné, le contexte $\mathcal{K}_i$ auquel il appartient.

Définition 43. *Étant donné un contexte* $\mathcal{K}_i = (O_i, A_i, I_i)$, *la fonction ctx est*

définie comme suit :

$$\forall o \in O_i, ctx(o) = \mathcal{K}_i.$$

Un graphe d'objets associé à un objet donné o, noté $G(o)$, est composé d'un ensemble de nœuds, noté $V(o)$ représentant des objets de la FCR et d'un ensemble d'arcs, noté $E(o)$, représentant les liens inter-objets dans cette FCR. Ce graphe est construit par niveau en commençant par le premier niveau composé du nœud racine qui correspond à l'objet o et en ajoutant à chaque nouveau niveau les nouveaux nœuds qui correspondent aux objets liés aux objets du niveau précédent. La construction s'arrête au point de la saturation du graphe, c'est-à-dire, au point où aucun nouveau nœud ne peut être rajouté au graphe.

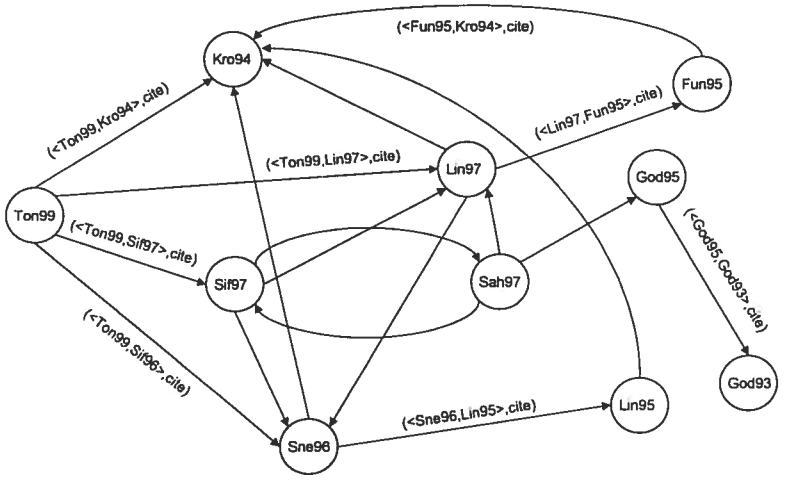


FIG. 4.8 – Le graphe de l'objet Ton99 obtenu à partir de la FCR composée du contexte des publications et de la relation cite.

Autrement dit, au départ, l'ensemble des nœuds du graphe, noté $V^0(o)$, est composé d'un seul nœud, celui qui correspond à l'objet o (la racine du graphe). Le graphe de niveau $i+1$ est obtenu par l'ensemble des nœuds (respectivement les arcs) du niveau i auxquels on rajoute tous les objets (nœuds) qui peuvent être atteints à partir des objets du niveau i par une quelconque relation de la FCR. L'ensemble des

nœuds du graphe obtenus après saturation est noté $V^{\infty}(o)$. La Figure 4.8 montre une partie du graphe d'objets de Ton99 du contexte des publications. Formellement, un graphe d'objets est défini de la manière suivante :

Définition 44. *Étant donné un objet o appartenant à O_i, le graphe d'objets correspondant, dénoté $G(o) = (V(o), E(o))$, est un multi-graphe orienté et étiqueté dont les nœuds sont des objets, les arcs représentent les liens relationnels et les étiquettes des arcs sont des noms de relations, tel que :*

$$
\begin{aligned}
&V^0(o) = \{o\},\\
&V^{i+1}(o) = V^i(o) \cup \bigcup\{r(\bar{o}) | \bar{o} \in V^i(o), r \in rel(ctx(\bar{o}))\},\\
&V(o) = V^{\infty}(o),\\
&E(o) = \{(\langle o_1, o_2\rangle, r) \mid o_1, o_2 \in V(o), \exists r \in \mathbf{R} : (o_1, o_2) \in r\}.
\end{aligned}
$$

Exemple : La Figure 4.9 montre les nouveaux nœuds ajoutés à chaque niveau au cours de la construction du graphe $G(Ton99)$. Cette figure illustre aussi l'évolution de l'ensemble de nœuds $V^i(Ton99)$. Le calcul de $V(Ton99)$ est comme suit :

$$
\begin{aligned}
&V^0(Ton99) = \{Ton99\},\\
&V^1(Ton99) = \{Ton99, Kro94, Lin97, Sif97, Sne96\},\\
&V^2(Ton99) = \{Ton99, Kro94, Lin97, Sif97, Sne96, Fun95, Sah97, Lin95\},\\
&V^3(Ton99) = \{Ton99, Kro94, Lin97, Sif97, Sne96, Fun95, Sah97, Lin95, God95\},\\
&V^4(Ton99) = \{Ton99, Kro94, Lin97, Sif97, Sne96, Fun95, Sah97, Lin95, God95, God93\},\\
&V^5(Ton99) = V^4(Ton99)\ \textit{niveau de saturation},\\
&V(Ton99) = V^{\infty}(Ton99) = V^4(Ton99).
\end{aligned}
$$

À noter qu'à tout objet $o \in \mathbf{O}$ ($\mathbf{O} = \{O_i | \mathcal{K}_i \in \mathbf{K}\}$), correspond un graphe d'objets dont la racine unique est le nœud qui représente cet objet. Pour déterminer l'ensemble des objets communs auxquels deux objets quelconque sont liés, il faut comparer les deux graphes d'objets respectifs et prendre les noeuds qui sont connectés aux racines par des chemins ayant les mêmes suites d'étiquettes (noms de relations). En effet, la racine est le point de départ pour un type particulier de chemins, dit profil, comme défini ci-après.

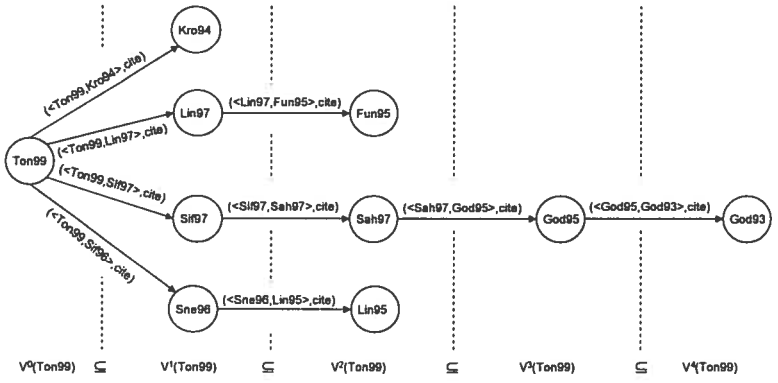


FIG. 4.9 – Une partie du graphe d'objets $G(Ton99)$ montrant les différents niveaux conduisant à la saturation de l'ensemble $V(Ton99)$.

4.7.2 Profil de chemin dans un graphe d'objets

Nous allons analyser les chemins partant d'un nœud représentant un objet donné o dans le graphe associé $G(o)$ et examiner les objets aux niveaux des nœuds d'arrivées. Pour ce faire, nous allons d'abord définir un profil pour un chemin dans un graphe d'objets. Ce profil représente tous les chemins ayant des séquences d'étiquettes similaires (noms de relations).

Formellement, un profil de chemin est une séquence de relations telles que les relations avoisinantes sont reliées au même contexte de manière complémentaire :

Définition 45 (Profil de chemin). *Étant donné une FCR* $(\mathbf{K}, \mathbf{R})$ *où* $\mathbf{R} = \{r_1, \ldots, r_n\}$. *Un profil de chemin* P *dans les graphes associés aux objets des différents contextes de la FCR est défini comme suit :*

$$P = \langle r_1, r_2, \cdots, r_n \rangle \ : \ \begin{cases} \|P\| = n \\ \forall i \in [1, \ldots, n-1] \ dom(r_{i+1}) = cod(r_i) \end{cases}$$

Exemple : Dans le graphe d'objets de la Figure 4.10, on trouve les profils de chemins $\langle cite \rangle$ et $\langle cite, cite \rangle$ de longueur 1 et 2, respectivement. À partir de la Figure 6.8 qui montre les contextes et les relations qui composent une FCR encodant un modèle UML on peut définir beaucoup de profils avec des types de relations différents telles que ⟨ `owned_attribute,type,owned_operation`⟩ et ⟨

owned_assoc_end,belongs_to_assoc⟩. Ces relations illustrent des liens entre les éléments d'un modèle UML tels que les attributs, les classes, les rôles, etc.

Un ensemble de profils est noté $\prod$. Formellement, nous le définissons de la manière suivante :

Définition 46. *Étant donné un contexte $\mathcal{K}_i$, l'ensemble de tous les profils de chemins possibles P tel que $r_j \in rel(\mathcal{K}_i)$ est dénoté $\prod_i$.*

4.7.3 Instance de profil

Un profil peut avoir des instances dans un graphe d'objets. Nous définissons une instance d'un profil P comme étant l'ensemble de tous les objets au niveau des nœuds d'arrivée des chemins ayant la même séquence d'étiquettes que le profil.

Formellement, considérons P un profil de taille n, l'instance $P(o)$ est définie de la manière suivante :

Définition 47. *Étant donné un profil $P = \langle r_1, r_2, \cdots, r_n \rangle$ et un objet o, alors :*
$P(o) = P_{[n]}(o)$ *where* $\begin{cases} P_{[0]}(o) = \{o\} \\ P_{[i+1]}(o) = \bigcup\{r_{i+1}(\bar{o}) | \bar{o} \in P_{[i]}(o)\} \end{cases}$

Évidemment, si $ctx(o) \neq dom(r_1)$, alors le profil ne peut être instancié à partir de cet objet. Les instances de chemin sont en fait une généralisation de la notion de lien relationnel du moment qu'elles permettent l'enchaînement de liens.

Exemple : Étant donné le profil de chemin $P = \langle cite \rangle$ et l'objet Kui00 du contexte $\mathcal{K}_p$, on a : P(Kui00)=P_1(Kui00)={Sne96, Sif97, Lin97, van98, Sne98C, Sne98R, Sne00S}. Considérons maintenant le profil de chemin $P = \langle cite, cite \rangle$ et le même objet Kui00 du contexte $\mathcal{K}_p$, on a : P(Kui00)=P_2(Kui00)={God93, Sne96, Sif97, Lin95, van98, Sne98C, Sne98R, Sne99, Fun95, Kro94, Lin97, Sah97}.

4.7.4 Propriété de formation de concepts

Afin de formuler la propriété reliant les treillis finaux obtenus par le processus itératif de l'ARC aux données de départ encodées dans la FCR, il faut rappeler

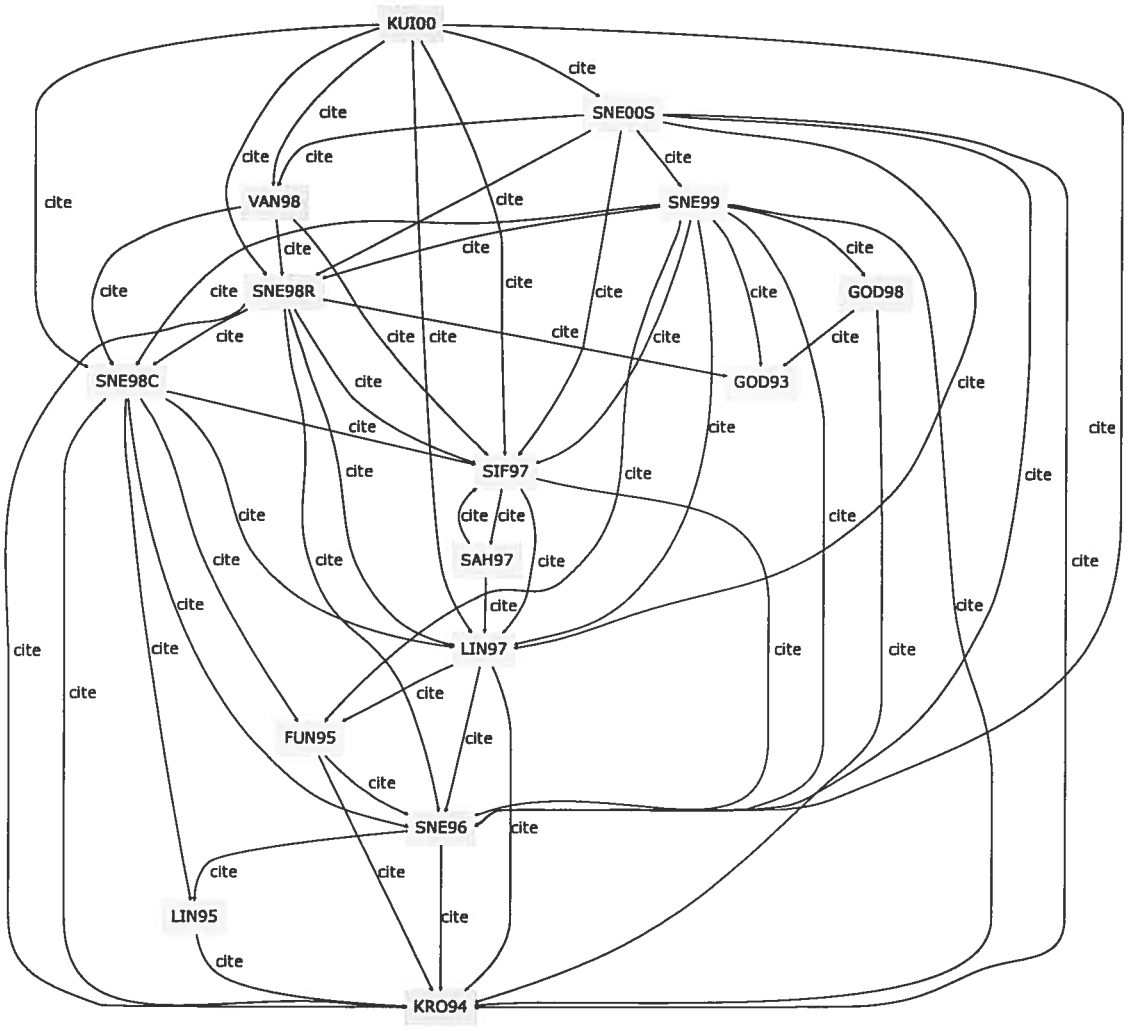


FIG. 4.10 – Une partie du graphe d'objets obtenu à partir de la FCR composée du contexte des publications et de la relation cite.

qu'à une étape donnée de ce processus, un concept c émerge dans un treillis $\mathcal{L}_i$ associé au contexte $\mathcal{K}_i$ pour deux raisons non exclusives :

- Les objets dans $Ext(c)$ partagent des attributs initiaux ($Int_l(c) \neq \emptyset$)
- Les objets dans $Ext(c)$ partagent des attributs relationnels ($Int_r(c) \neq \emptyset$).

De plus, même si deux objets formels donnés o_1 et o_2 ne partagent aucun attribut initial ($o_1' \cap o_2' = \emptyset$ et donc ne forment aucun concept dans le treillis initial), ils peuvent se retrouver dans l'extension d'un même concept dans le treillis final autre que le concept top ($\top$). En effet, si à première vue o_1 et o_2 ne partagent aucun attribut, les objets auquels ils sont tous deux reliés peuvent avoir des attributs similaires, notamment des attributs initiaux (de A_i^0).

Exemple : Les deux objets Kui00 (objet 8) et Til03S (objet 21) dont les concepts objets respectifs dans le treillis initial du contexte des publications de la Figure 4.1 sont $c_{\#3}$ et $c_{\#0}$, respectivement, ne partagent aucun attribut initial et par conséquent ils sont présents uniquement au niveau de l'extension de top (concept $c_{\#2}$). Par contre dans le treillis de la première itération illustré par la Figure 4.3, ces deux objets apparaissent déjà au niveau du concept $c_{\#2}$ ($\top$) et $c_{\#8}$. Au niveau du concept $c_{\#8}$ de la Figure 4.3, Kui00 et Til03S partagent les attributs {`cite:0`, `cite:2`, `cite:4`} ce qui signifie que les deux objets sont liés aux objets qui possèdent des caractéristiques initiales similaires (voir l'intension du concept `cite:0`). Dans le treillis final de la Figure 4.4, ces deux objets apparaissent dans les extensions des concepts $c_{\#2}$ ($\top$), $c_{\#8}$ et $c_{\#10}$. En examinant le concept $c_{\#10}$ par exemple, on constate que les deux objets Kui00 (8) et Til03S (21) partagent l'ensemble d'attributs {`cite:0`, `cite:2`, `cite:4`, `cite:7`, `cite:8`}. Cela signifie aussi que Kui00 et Til03S sont reliés à des objets qui partagent non seulement des attributs initiaux mais aussi relationnels (voir l'intension du concept `cite:7`).

Pour revenir au cas général, les deux objets o_1 et o_2, apparaissent dans l'extension d'un concept autre que le concept $\top$ si et seulement s'il existe un profil P dans les graphes d'objets associés et dont les instances respectives $P(o_1)$ et $P(o_2)$ partagent un attribut initial. C'est ce que exprime la propriété suivante :

Propriété 20. *Étant donné deux objets o_1 et o_2 du contexte $\mathcal{K}_i$. $o_1, o_2 \in Ext(c)$*

où c est un concept du treillis $\mathcal{L}_i$ si et seulement s'il existe un profil P tel que :

- *cas d'un codage large :* $\cup_{o \in P(o_1)} o' \bigcap \cup_{o \in P(o_2)} o' \neq \emptyset$
- *cas d'un codage étroit :* $\cap_{o \in P(o_1)} o' \bigcap \cap_{o \in P(o_2)} o' \neq \emptyset$

Ainsi, dans le cas d'un codage large, pour que les deux objets o_1 et o_2 se retrouvent ensemble dans l'extension d'un même concept, il suffit qu'il existe deux objets, l'un dans l'instance de profil $P(o_1)$ et l'autre dans l'instance $P(o_2)$ qui partagent un attribut. Dans le cas d'un encodage étroit, il faut absolument que tous les objets de l'instance de profil $P(o_1)$ partagent au moins un attribut avec tous les objets de l'instance $P(o_2)$.

Exemple : Considérant les profils de chemins $\langle cite \rangle$ et $\langle cite, cite \rangle$. Les instances de ces profils relatives aux deux objets Kui00 et Til03S du contexte $\mathcal{K}_p$ sont :

- P_1(Til03S)={Lin97, Sne98R},
- P_1(Kui00)={Sne96, Sif97, Lin97, van98, Sne98C, Sne98R, Sne00S},
- P_2(Til03S)={Fun95, Kro94, Sne96, God93, Lin97, Sif97, Sne98C },
- P_2(Kui00)={God93, Sne96, Sif97, Lin95, van98, Sne98C, Sne98R, Sne99, Fun95, Kro94, Lin97, Sah97}.

Bien que les objets Kui00 et Til03S ne partagent aucun attribut, ils se retrouvent dans l'extension du concept $c_{\#8}$ de la Figure 4.3 car les objets dans les deux instances du profil $\langle cite \rangle$ relatives à ces deux objets ont en commun les attributs initiaux suivants :

$$\cap_{o \in P_1(Kui00)} o' \bigcap \cap_{o \in P_1(Til03S)} o' = \{cd, ma\}.$$

À noter que ce sous ensemble d'attributs constitue l'intension du concept $c_{\#0}$ du treillis obtenu avec un scaling étroit et illustré par la Figure 4.3. Ceci explique la présence de la relation cite dans la Figure 4.3 entre le concept $c_{\#8}$, le concept-objet de Kui00 et Til03S, et le concept $c_{\#0}$ (ainsi que tous ses successeurs selon la propriété 19) car cette relation inter-concepts constitue une abstraction des liens inter-objets entre, d'une part, les objets Kui00 et Til03S (concept $c_{\#8}$) et d'autre part les objets appartenant aux instances de profil $P = \langle cite \rangle$ respectifs ($c_{\#0}$).

Considérons maintenant le profil de chemin $P = \langle cite, cite \rangle$ et les mêmes objets Kui00 et Til03S. Les objets dans les deux instances de ce profil relatives aux objets Kui00 et Til03S partagent le sous ensemble suivant :

$$\cap_{o \in P_2(Kui00)} o' \bigcap \cap_{o \in P_2(Til03S)} o' = \cap_{o \in P_1(Kui00)} o' \bigcap \cap_{o \in P_1(Til03S)} o' = \{cd, ma\}.$$

Selon le treillis final des publications $\mathcal{L}_p^\infty$ obtenu par un scaling étroit et illustré par la Figure 4.4, cet sous ensemble d'attributs appartient aux intensions des concepts $c_{\#0}$, $c_{\#7}$ et $c_{\#9}$. Ceci explique aussi la présence de la relation cite dans la Figure 4.4 entre le concept $c_{\#10}$, le concept-objet de Kui00 et Til03S, et le concept $c_{\#7}$, le concept-attribut des attributs partagés par les instances de profil des deux objets.

Finalement, les concepts d'un treillis final $\mathcal{L}^\infty$ contiennent toutes les possibilités de partage de structure entre objets. Cela est vraie même si les ensembles des objets reliés et partageant des attributs initiaux peuvent être situés très loin dans les graphes d'objets respectifs.

4.8 ARC et logiques de descriptions

L'ARC produit des abstractions relationnelles dont l'exploitation nécessite un outil permettant des opérations telles que la modification, la confrontation, la vérification de consistance, etc. L'utilisation d'un raisonneur[8] LD est une solution de choix car :

- La complémentarité que nous avons étudié entre l'AFC et les systèmes de la LD facilite l'expression des résultats de l'ARC en une logique de descriptions (voir Chapitre 3, section 3.1.1.5),
- Les divers services d'inférences qu'un système à base de connaissances LD peut effectuer permettent l'exploitation de ces résulats.

[8]Tels que RACER (Renamed ABox and Concept Expression Reasoner) [37] et FACT (Fast Classification of Terminologies) [38].

Dans cette section nous allons étudier la complémentarité entre l'ARC et les logiques de descriptions. En effet, nous allons décrire comment les données de l'ARC ainsi que les résultats issus de la méthode MULTI-FCA sont exportés en une base de connaissances en utilisant un langage LD.

4.8.1 Choix d'un language LD

La création de concepts non primitifs en logiques de descriptions nécessite forcément l'opérateur de définition de concepts $\equiv$. D'autre part, les descriptions des individus dans la FCR sont sous la forme de conjonctions de caractéristiques d'où le besoin de l'opérateur de conjonction logique $\sqcap$. De plus, la méthode de l'ARC enrichit les contextes de la FCR par l'ajout d'attributs relationnels ayant la forme $r : c$ qui signifie, rappelons le, que tous les liens de type r vont vers les individus dans l'extension du concept c. Ceci s'interprète en logiques de descriptions par une restriction du rôle qui correspond à la relation r aux individus appartenant à l'interprétation du concept logique qui correspond au concept formel c, d'où le besoin de l'opérateur de restriction de valeur $\forall$. De ce fait, on constate que le langage LD le plus simple offrant ces opérateurs et capable d'exprimer toute la connaissance que renferme un treillis relationnel est le langage $\mathcal{FL}_0$ décrit dans le Chapitre 3, Section 3.1.1.2.1.

4.8.2 Espace de noms d'une FCR

A fin de faciliter le raisonnement sur les éléments de la FCR notamment les contextes, objets formels, attributs formels et relations, nous allons associer à une FCR un seul espace de noms dans lequel tout élément a un nom unique. De plus, les contextes sont déterminés par leurs ensembles d'objets. Ces ensembles sont invariants tout au long du processus d'analyse de l'ARC. De façon similaire, un concept formel est défini uniquement par son extension qui, tout comme les ensembles d'objets des différents contextes, s'avère invariante à travers le mécanisme de scaling relationnel. En effet, comme il est démontré dans [91, 55], l'ajout d'attributs for-

mels à un contexte engendre une expansion (potentielle) du treillis correspondant. En effet, le treillis augmenté contient le treillis initial car en comparant les deux treillis suivant les extensions des concepts on constate l'existence de trois catégories de concepts, à savoir des nouveaux, des inchangés et des modifiés (voir [86] et [56] pour plus de détails).

4.8.3 Construction de la base de connaissances

Nous construisons une base de connaissances, dénotée $\mathcal{B} = (\mathcal{T}(\mathcal{T}_C, \mathcal{T}_R), \mathcal{A})$, en utilisant le language $\mathcal{FL}_0$, à partir d'une FCR $(\mathbf{K}, \mathbf{R})$ et de l'ensemble des treillis finaux $\mathcal{L}_i^\infty$ des différents contextes de la manière suivante :

4.8.3.1 Génération de la terminologie (TBox)

D'abord, l'ensemble des symboles de $\mathcal{R}$, à savoir, les noms de contextes, d'attributs, d'objets et de relations sont envoyés à $\mathcal{B}$ par le biais d'une fonction générique, dénotée ι, qui constitue une bijection. La fonction ι établit une correspondance entre, d'une part les concepts objets formels, attributs formels et concepts des treillis et d'autre part les concepts logiques atomiques dans $\mathcal{B}$. Formellement, elle est décrite de la manière suivante :

- $\forall \mathcal{K}_i \in \mathbf{K}\ \iota[\mathcal{K}_i] \in \mathcal{T}_C$,
- $\forall a_{i,j} \in A_i^0\ \iota[a_{i,j}] \in \mathcal{T}_C$,
- $\forall o_{i,j} \in O_i\ \iota[o_{i,j}] \in const(\mathcal{A})$, (constantes de la ABox),
- $\forall r_i \in \mathbf{R}\ \iota[r_i] \in \mathcal{T}_R$,
- $\forall c_{i,j} \in \mathcal{L}_i^\infty\ \iota[c_{i,j}] \in macros(\mathcal{T})$, (symboles de la TBox).

Ceci signifie que les contextes et les attributs deviennent des concepts primitifs alors que les relations deviennent des rôles. Les objets sont traduits en individus (constantes). La fonction ι permet principalement d'établir la terminologie (TBox) de la base de connaissance par la définition des noms des concepts logiques ainsi que de générer un ensemble de faits de départ (ABox). De cette manière, elle permet la création effective du contenu de la base de connaissances.

Exemple : La Figure 4.11 illustre le contenu de la TBox au bout de la première étape de l'exportation des données du contexte des publications $\mathcal{K}_p$ (Table 4.1), de la relation cite (Table 4.2) et du treillis final $\mathcal{L}_p^{\infty}$ (Figure 4.4).

```
(in-knowledge-base PAPER-RCF)
(signature :atomic-concepts( PAPER AB CD CA MA PAPER-C2 PAPER-C4 PAPER-C5
PAPER-C8 PAPER-C0 PAPER-C6 PAPER-C3 PAPER-C7 PAPER-C10 PAPER-C1 PAPER-C9)
:roles((cite))
:individuals(Fun95 God93 God95 God98 Huc99 Huc02 Kro94 Kui00 Leb99 Lin95
Lin97 Sah97 Sif97 Sne00S Sne00U Sne96 Sne98C Sne98R Sne99 Str99 Til03S
Til03T Ton99 Tone01 Van98))
```

FIG. 4.11 – Résultat de la première étape de génération de la TBox à partir du contexte des publications $\mathcal{K}_p$ et de son treillis $\mathcal{L}_p^{\infty}$ illustré par la Figure 4.4.

La deuxième étape du processus de génération de la TBox consiste à traduire les attributs relationnels et les concepts des treillis finaux obtenus par la méthode de l'ARC. Ainsi, tout attribut relationnel est traduit en restriction de rôle et tout concept d'un treillis final est traduit en une expression logique de la manière suivante :

- $\forall a_{i,j} = r_l : c_{m,n} \in A^{\infty} - A_i^0$, $\iota[a_{i,j}] = \forall \iota[r_l].\iota[c_{m,n}] \in \mathcal{T}$,
- $\forall c_{i,j} \in \mathcal{L}_i^{\infty}$, $(\iota[c_{i,j}] \equiv \sqcap_{a_{i,j} \in int(c_{i,j})} \iota[a_{i,j}]) \in \mathcal{T}$,

La Figure 4.12 montre les nouvelles expressions terminologiques ajoutées à la TBox de la Figure 4.11 .

4.8.3.2 Génération des faits (ABox)

Dans une seconde étape de l'exportation d'une FCR en une base de connaissances en LD, les faits de la ABox sont déterminés. Ainsi, toutes les incidences objet-attribut dans I_i^0, les incidences objet-contexte, les liens inter-objets et l'appartenance des objets aux différentes extensions de concepts sont traduits en fait dans la ABox. Formellement, la fonction ι

```
(implies PAPER PAPER)
(implies PAPER-C4 CD)
(implies PAPER-C5 CA)
(implies PAPER-C8 (all cite PAPER-C0)))
(implies PAPER-C0 (and MA PAPER-C4))
(implies PAPER-C6 (and PAPER-C5 PAPER-C4))
(implies PAPER-C3 (and AB (and PAPER-10 PAPER-C5)))
(implies PAPER-C7 (and PAPER-C8 PAPER-C0))
(implies PAPER-C10 (and (all cite PAPER-C7) PAPER-C8))
(implies PAPER-C1 (and PAPER-C3 (and PAPER-C9 PAPER-C6)))
(implies PAPER-C9 (and PAPER-C10 PAPER-C7))
```

FIG. 4.12 – Résultat de la deuxième étape de génération de la TBox à partir du treillis $\mathcal{L}_p^\infty$ illustré par la Figure 4.4.

- $\forall a_{i,j} \in A_i^0, \forall o_{i,j} \in a'_{i,j}$, $\iota[a_{i,j}](\iota[o_{i,j}]) \in \mathcal{A}$,
- $\forall o_{i,j} \in O_i$, $\iota[\mathcal{K}_i](\iota[o_{i,j}]) \in \mathcal{A}$,
- $\forall r_i \in \mathbf{R}, \forall o_1, o_2 : r(o_1, o_2)$, $\iota[r_i](\iota[o_1], \iota[o_1]) \in \mathcal{A}$,
- $\forall c_{i,j} \in \mathcal{L}_i^\infty, \forall o_{i,j} \in ext(c_{i,j})$, $\iota[c_{i,j}](\iota[o_{i,j}]) \in \mathcal{A}$,

Exemple : La Figure 4.13 illustre le contenu de la ABox qui correspond à la TBox illustrée par la Figure 4.11 et Figure 4.12.

```
(instance Fun95 PAPER)
(instance God93 PAPER)
...
(related Fun95 Kro94 cite)
(related Huc99 God93 cite)
...
```

FIG. 4.13 – ABox qui correspond au treillis $\mathcal{L}_p^\infty$ illustré par la Figure 4.4.

4.8.4 Propriété de fermeture des descriptions dans la TBox

Afin d'expliciter les correspondances entre les faits dans la base de connaissances $\mathcal{A}$ et la terminologie associée $\mathcal{T}$, nous considérons la ABox $\mathcal{A}$ comme le co-domaine de la fonction d'interprétation $\mathcal{I}$. Cela signifie que toutes les descriptions de concepts et de rôles dans $\mathcal{T}$ sont interprétées en terme d'individus et de paires d'individus, respectivement, de $\mathcal{A}$. À noter que nous disposons de la sémantique de tous les concepts définis dans la TBox $\mathcal{T}$ car toutes les extensions des concepts formels des treillis finaux $\mathcal{L}_i^{\infty}$ ont été explicitement traduites en faits dans la ABox $\mathcal{A}$.

On cherche à savoir si toutes les similarités dans la description des individus ont été exprimées par le langage $\mathcal{FL}_0(\mathcal{T}(\mathcal{T}_C, \mathcal{T}_R)$ ou bien il reste encore d'autres partages de descriptions entre individus qui n'ont pas été identifiés par la méthode de l'ARC et qui pourraient encore être admis. Dans ce sens, nous formulons la propriété suivante :

Propriété 21 (Fermeture des descriptions). *Étant donné une description arbitraire et non circulaire D de $\mathcal{FL}_0(\mathcal{T}(\mathcal{T}_C, \mathcal{T}_R)$, il existe un concept C de $\mathcal{T}(\mathcal{T}_C, \mathcal{T}_R)$ tel que D et C sont sémantiquement équivalents par rapport au modèle $\mathcal{A}$ ($D^{\mathcal{A}} \equiv C^{\mathcal{A}}$).*

La Propriété 21 s'avère équivalente à la Propriété 19 en terme de logiques de descriptions. Elle précise que, sous certaines hypothèses, l'ensemble de toutes les sémantiques possibles, c'est-à-dire, les sous-ensembles d'objets qui peuvent être décrits par des formules du langage $\mathcal{FL}_0(\mathcal{T}(\mathcal{T}_C, \mathcal{T}_R)$, sont déjà dans la TBox $\mathcal{T}$ obtenue par la traduction des résultats de la méthode de l'ARC.

4.8.5 Simplification des attributs relationnels

Les opérateurs de codage (étroit ou large) du mécanisme de scaling de la méthode de l'ARC produisent des interprétations redondantes des attributs relationnels. En effet, la présence d'un lien vers un concept donné signifie aussi des liens avec tous ses successeurs dans le même treillis. Ainsi, dans le cas de l'existence

de plusieurs liens dans une même intension, il suffit de garder le lien avec le concept minimal.

À noter que l'élimination des liens dits "redondants" produit un treillis lisible vu le contenu moins volumineux des intensions des concepts. De plus, elle permet de produire des descriptions compactes de concepts logiques lors de l'exportation d'un treillis relationnel en une base de connaissance en LD. Cette simplification des intensions se base sur la propriété suivante :

Propriété 22. *Étant donné un concept relationnel* $c = (X, Y)$, *tel que* $rel(c)=r : c_1 r : c_2$ *et* $c_1 \leq_K c_2$. *Par conséquent, la partie relationnelle de l'intension de c peut être comme suit :* $rel(c)=r : c_1$.

Démonstration. $C \equiv \sqcap A_i \ \sqcap \forall R \cdot C_1 \ \sqcap \forall R \cdot C_2$.
Si $C_1 \sqsubseteq C_2$ alors $\forall R \cdot C_1 \sqcap \forall R \cdot C_2 \equiv \forall R \cdot C_1$ d'où $C \equiv \sqcap A_i \sqcap \forall R \cdot C_1$.
Finalement, $rel(c)=r : c_1$. □

4.9 Conclusion

Nous avons montré comment les échelles relationnelles et le mécanisme de scaling relationnel permettent de traiter les liens inter-objets dans une famille de contextes. La méthode proposée a conduit à la définition d'intensions de concepts qui incluent des relations avec d'autres concepts décrivant, à l'instar des associations en UML et des rôle en logiques de descriptions, de manière générique les liens existant entre les objets de l'extension d'un tel concept et d'autres objets formels. Nous avons aussi expliqué en terme de graphes d'objets la formation des concepts relationnels. Dans le chapitre suivant nous allons mettre au point une adaptation de la méthode de calcul de concepts relationnels (MULTI-FCA) aux treillis complets en utilisant les types de scaling relationnel définis dans ce chapitre.

Chapitre 5

Cadre algorithmique de l'ARC

Nous décrivons ici les aspects algorithmiques de l'ARC. Nous allons présenter le schéma général de la méthode de construction des treillis relationnels à partir d'une FCR, dite "MULTI-FCA" ainsi qu'une instanciation algorithmique de ce schéma. Nous allons aussi décrire les principaux algorithmes auxiliaires, notamment l'algorithme de scaling relationnel et l'algorithme qui permet de maintenir un treillis par ajout d'attribut, appelé 'MAGALICE-A'.

5.1 Méthode Multi-FCA

La méthode MULTI-FCA implémente la stratégie de calcul itérative décrite dans le Chapitre 4, Section 4.6.1. Elle permet de traiter une FCR avec (ou sans) dépendances circulaires entre les différents contextes et produit une famille de structures conceptuelles cibles (treillis/SHG).

Le traitement commence par la construction des structures initiales des contextes de la FCR en utilisant les attributs formels. Ensuite, les descriptions des objets formels dans les différents contextes sont enrichies par des attributs reflétant les abstractions sur les co-domaines des relations sous-jacentes. Puis, les structures conceptuelles sont mises à jour en conséquence. Autrement dit, les liens inter-objets exprimés par les différentes relations de la FCR sont utilisés pour augmenter les

descriptions des objets en plusieurs étapes. À chaque nouvelle étape, des échelles relationnelles sont construites à partir des treillis de l'étape précédente et utilisées pour enrichir les contextes par de nouveaux attributs formels. Ces attributs formels reflètent les liens inter-objets au sein du contexte. Comme les résultats d'une étape conduisent à la modification des contextes de la FCR, les treillis respectifs et par là les échelles relationnelles qui en sont issues doivent aussi être modifiées et donc une étape supplémentaire peut être accomplie. Ce processus itératif s'arrête dès que les nouveaux treillis sont isomorphes aux treillis actuels. On dit aussi que les contextes sont saturés et qu'aucun enrichissement n'est possible. Dans ce qui suit, nous allons décrire notre méthode.

5.1.1 Schéma général de Multi-FCA

L'algorithme 2 illustre le schéma général de la méthode MULTI-FCA. Le principe de l'algorithme consiste d'abord, à trouver une solution initiale du problème de construction des structures cibles (treillis/SHG correspondants à la FCR), ensuite, raffiner à chaque nouvelle étape cette solution en utilisant les résultats de l'étape précédente. Le résultat final est 'le point fixe' de l'opération de construction des structures cibles.

La solution initiale consiste à faire un éventuel scaling conceptuel des attributs pluri-valués (primitive SCALE-BIN, *ligne* 5) et à construire les structures cibles qui correspondent aux différents contextes de la FCR en utilisant une méthode '*classique*' de construction à partir d'un contexte binaire (primitive FCA, *ligne* 6). Par la suite, la méthode itère sur l'ensemble des contextes. Pour un contexte donné, une itération se déroule en deux étapes. Tout d'abord, la méthode calcule l'extension relationnelle du contexte par le scaling relationnel des différentes relations en utilisant les structures de l'itération courante comme des échelles (primitive EXTEND-REL, *ligne* 10). Suite à cette étape, les descriptions des objets appartenant aux différents contextes sont étendues et raffinées. L'étape suivante permet de mettre à jour la structure conceptuelle respective (primitive FCA, *ligne* 11).

Le processus itératif s'arrête dès que toutes les structures obtenues à l'itération en cours restent isomorphes à celles de l'itération précédente. À ce moment, on dit que les différents contextes de la FCR sont saturés.

```
1: proc MULTI-FCA( In : (K, R) a RCF,
2:                 Out : S array [1..n] of identical conceptual structures)
3: p ← 0 ; halt ← false
4: for i from 1 to n do
5:    K_i^p ← SCALE-BIN(K_i)
6:    S^p[i] ← BUILD-CONCEPTUAL-STRUCT(K_i^p)
7: while not halt do
8:    p++
9:    for i from 1 to n do
10:      K_i^p ← EXTEND-REL(K_i^{p-1}, S^{p-1})
11:      S^p[i] ← BUILD-CONCEPTUAL-STRUCT(K_i^p)
12:   halt ← ⋀_{i=1,n}(S^p[i] = S^{p-1}[i])
```

Algorithm 2: Construction d'une famille de structures (treillis/SHG) relationnels.

5.1.2 Instanciation du schéma général de Multi-FCA

La méthode MULTI-FCA décrite par l'algorithme 2 peut être instanciée de différentes manières dépendamment de la structure cible, des structures de données utilisées et du mode de scaling relationnel que l'on veut avoir. L'algorithme 3 présente une instanciation du schéma de cette méthode. Il permet de calculer une famille de treillis relationnels (FTR) à partir d'une FCR en appliquant un mode de scaling paramétrable et en s'appuyant sur un ensemble de routines auxiliaires.

5.1.2.1 Principe

L'algorithme 3 suit le schéma général donné par l'algorithme 2. En effet, il commence par l'initialisation des structures cibles aux treillis initiaux ; ensuite il alterne, aussi longtemps que les contextes ne sont pas saturés, entre l'extension

relationnelle des contextes et la mise à jour des treillis respectifs. Au cours d'une même itération et entre le calcul de l'extension relationnelle de deux contextes différents, un treillis peut évoluer entre le calcul de l'extension relationnelle d'un contexte donné et le calcul de l'extension relationnelle d'un autre contexte. Par exemple, soient les trois contextes $\mathcal{K}_1 = (O_1, A_1, I_1)$, $\mathcal{K}_2 = (O_2, A_2, I_2)$ et $\mathcal{K}_3 = (O_3, A_3, I_3)$ et trois relations $r_1 : O_1 \rightarrow 2^{O_2}$, $r_2 : O_2 \rightarrow 2^{O_1}$ et $r_3 : O_3 \rightarrow 2^{O_2}$. Supposons que l'ordre de l'extension relationnelle des contextes (Algorithme 3, ligne 22) est $\mathcal{K}_1$, $\mathcal{K}_2$ et $\mathcal{K}_3$. Il est évident qu'à une étape donnée du processus itératif le treillis $\mathcal{L}_2$ qui servira au scaling de la relation r_1 n'est pas forcement le même treillis qui servira au scaling de la relation r_3. Afin de permettre que les mêmes concepts d'un treillis soient utilisés pour le scaling de toutes les relations dépendantes, le processus du scaling utilise le même ensemble $\Delta\mathbf{L}$ au sein d'une itération donnée pour le scaling de toutes les relations dépendantes.

5.1.2.2 Structures de données utilisées

L'algorithme 3 utilise divers vecteurs pour stocker les données issues de deux itérations consécutives. Le vecteur $\mathbf{L}$ et $\mathbf{L}^+$ représentent les treillis actuels et ceux obtenus au bout de l'itération en cours, respectivement. Les concepts qui serviront à la création des attributs relationnels sont rangés dans l'ensemble $\Delta\mathbf{L}$. À la fin de toute itération, de nouveaux concepts $\Delta\mathbf{L}^+$ sont calculés pour chaque treillis et serviront à un nouveau tour de scaling des différentes relations. Finalement, à chaque contexte correspond une position dans le vecteur booléen `Off` qui indique si le contexte est déjà saturé ou pas encore.

5.1.2.3 Description détaillée de l'algorithme

L'algorithme 3 commence par le scaling des attributs pluri-valués des différents contextes (ligne 11) dans le but de calculer les treillis initiaux (ligne 13). 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 aux concepts actuels des

treillis (ligne 14). L'algorithme termine l'étape d'initialisation par positionner les valeurs du vecteur `Off` à `faux` (ligne 15) car les contextes ne sont pas encore saturés. 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 (ligne17), l'algorithme initialise la variable de saturation globale (ligne 18). 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 $20-21$). Arrivé à ce stade, l'algorithme parcourt tous les contextes de la FCR (ligne 22) et examine leur état de saturation (ligne 23). Pour tout contexte non saturé, l'algorithme détermine l'ensemble de ses relations (ligne 24). Ensuite, pour chaque relation r de cet ensemble, l'algorithme réalise les trois opérations suivantes :

1. Calcul d'une nouvelle extension relationnelle du contexte suivant r (ligne 26) en utilisant le treillis du co-domaine de r obtenu au niveau de l'itération précédente ($\Delta\mathbf{L}$),
2. Mise à jour du contexte courant (ligne 27),
3. Mise à jour des treillis du contexte courant suite à l'évolution du contexte associé (ligne 28).

Par la suite, l'ensemble des nouveaux concepts qui ont émergé dans le treillis suite à l'enrichissement du contexte correspondant est calculé (ligne 29), d'une part, pour savoir si le contexte est saturé ou pas et, d'autre part, pour être utilisé dans de futures scaling. Si cet ensemble s'avère vide (ligne 30), cela signifie que le contexte est saturé et qu'il doit être ignoré au cours des prochaines itérations (ligne 31). Dans le cas ou au moins un treillis a évolué, le point fixe n'est pas atteint (ligne 33) obligeant l'algorithme à effectuer d'autres itérations. En fin de calcul du point fixe, l'algorithme retourne la FTR construite.

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

Algorithm 3: Traitement d'une FCR par la méthode MULTI-FCA.

5.1.2.4 Routines auxilliaires

L'algorithme 3 s'appuie sur plusieurs routines. La Table 5.1 récapitule le service fourni par chacune d'entre elles.

L'algorithme 4 illustre la manière avec laquelle le scaling relationnel d'une relation donnée r est effectué en utilisant un ensemble de concepts, noté $\Delta\mathcal{L}_j$, appartenant au treillis $\mathcal{L}_j$ du contexte co-domaine de r. L'algorithme construit donc une relation binaire, notée $\Delta\mathcal{K}_i$ qui servira à l'extension du contexte domaine de r, noté $\mathcal{K}_i$.

Pour ce faire, l'algorithme commence par la définition des objets formels de l'extension (ligne 5) ensuite pour chaque paire composée d'un individu o du contexte $\mathcal{K}_i$ et d'un concept c du treillis $\mathcal{L}_j$ (lignes 6-7), l'algorithme vérifie si l'image de l'individu o par la relation r est reliée d'une certaine façon à l'extension du concept c (lignes 9-15). À noter qu'au départ, l'algorithme suppose qu'il n'y a aucune liaison entre ces deux éléments (ligne 8). De plus cette liaison dépend du mode de scaling sollicité, à savoir, étroit ou large (voir Chapitre 4, Section 4.5.1.2). Dans le cas où le test de liaison s'avère positif (ligne 16), l'algorithme exécute un ensemble de tâches dont le but est l'intégration d'un nouvel attribut relationnel à l'extension (lignes 17-20). Ainsi, l'algorithme fait intervenir la routine CREATE qui se charge de retourner un nouveau attribut de la forme r:c (ligne 17). Ensuite, il vérifie si un tel attribut n'existe pas déjà parmi les attributs formels $\Delta\mathcal{A}_i$ de l'extension (ligne 18). Si l'attribut r:c n'est pas dans $\Delta\mathcal{A}_i$, il est ajouté (ligne 19) ensuite l'occurrence $o \times \{\texttt{r:c}\}$ est ajouté à la relation binaire $\Delta\mathcal{I}_i$ (ligne 20). En fin de traitement, l'algorithme retourne l'extension $\Delta\mathcal{K}_i$ créée (ligne 21). À noter que les SHG se construisent de façon identique.

5.2 Exemple de déroulement de Multi-FCA

Supposons maintenant que l'on veut appliquer la méthode MULTI-FCA (Algorithme 3) à la FCR composée du contexte 'humain', noté $\mathcal{K}_h$ et de la relation

Routine	*Tâche*	*Parametres*
CONCEPTUAL-SCALING	Calcule le contexte dérivé à partir du contexte pluri-valué.	**Entrée** : Un contexte pluri-valué $\mathcal{K}_i$. **Sortie** : le contexte dérivé $\mathcal{K}_i^d$.
COMPUTE-INITIAL-LATTICE	Construit le treillis de concepts.	**Entrée** : Un contexte $\mathcal{K}_i$. **Sortie** : le treillis $\mathcal{L}_i$.
GET-CONCEPTS	Réduit un treillis de concepts en un ensemble de concepts.	**Entrée** : un treillis de concepts. **Sortie** : un ensemble de concepts.
GET-CONTEXT-RELATIONS	Calcul l'ensemble des relations d'un contexte donné.	**Entrée** : un contexte $\mathcal{K}_i$ et l'ensemble des relations de la FCR **R**. **Sortie** : $rel(\mathcal{K}_i)$.
RANK	Localise un contexte donné dans le vecteur des contextes de la FCR.	**Entrée** :. Le vecteur des contextes **K** et le contexte co-domaine d'une relation $\mathcal{K}_j$. **Sortie** : l'indice de $\mathcal{K}_j$ (valeur $1..n$).
GET-RELATIONAL-SCALING	Effectue le scaling relationnel en s'appuyant sur l'algorithme décrit ci-dessous (Algorithme 4).	**Entrée** : un contexte $\mathcal{K}_i$, une relation r et le treillis du contexte co-domaine de r. **Sortie** : un contexte binaire.
MERGE	Fusionne deux contextes binaires.	**Entrée** : un contexte $\mathcal{K}_i = (O_i, A_i, I_i)$ et son extension relationnelle $\Delta\mathcal{K}_i = (O_i, A_i^r, I_i^r)$. **Sortie** : $\mathcal{K}_i = (O_i, A_i \cup A_i^r, I_i \cup I_i^r\})$.
UPDATE	Mis à jour d'un treillis par ajout d'attribut. Elle s'appuie sur l'algorithme MAGALICE-A([56]).	**Entrée** : un treillis et un ensemble de nouveaux attributs. **Sortie** : le treillis à jour.
GET-NEW-CONCEPTS	Retourne les concepts qui ont émergés dans un treillis donné au bout d'une étape du calcul itératif.	**Entrée** : deux versions consécutives d'un treillis. **Sortie** : la différence de concepts entre les deux versions.

TAB. 5.1 – Routines de l'algorithme 3.

1: **Algorithm** GET-RELATIONAL-EXTENSION
2: (**In :** $\mathcal{K}_i = (O_i, A_i, I_i)$ a source context, $r = O_i \times O_j$ an inter-objet binary relation, $\Delta\mathcal{L}_j$ a set of concepts, mode : scaling mode) : binary context
3: Local : has boolean variable
4: Local : $\Delta\mathcal{K}_i = (\Delta O_i, \Delta A_i, \Delta I_i)$ the relational extension of $\mathcal{K}_i$ along r
5: $\Delta O_i \leftarrow O_i$
6: **for all** $o \in O_i$ **do**
7: **for all** $c \in \Delta\mathcal{L}_j$ **do**
8: has $\leftarrow$ **false**
9: **switch**(mode)
10: case narrow :
11: **if** $r(o) \subseteq Ext(c)$
12: has $\leftarrow$ **true**
13: case wide :
14: **if** $r(o) \cap Ext(c) \neq \emptyset$
15: has $\leftarrow$ **true**
16: **if** has **then**
17: $a \leftarrow$ CREATE(r,c) {returns a new formal attribute having the form r:c}
18: **if** $a \notin \Delta A_i$ **then**
19: $\Delta A_i \leftarrow \Delta A_i \cup \{a\}$
20: $\Delta I_i \leftarrow \Delta I_i \cup \{(o, a)\}$
21: **return** $\Delta\mathcal{K}_i$

Algorithm 4: Calcul de l'extension relationnelle d'un contexte suivant une relation donnée.

'conjoint' ('spouse') de la Table 5.2. Les individus de ce contexte sont décrits par deux attributs pluri-valués, à savoir l'âge et l'expérience de travail. Tout individu est relié à un autre individu par la relation symétrique conjoint.

	o_1	o_2	o_3	o_4	o_5	o_6
Âge	25	28	22	26	33	35
E.t.	4	2	1	8	4	10

	o_1	o_2	o_3	o_4	o_5	o_6
o_1		X				
o_2	X					
o_3				X		
o_4			X			
o_5						X
o_6					X	

TAB. 5.2 – **Gauche :** le Contexte pluri-valué humain ($\mathcal{K}_h$). L'expression 'E.t.' signifie 'expérience de travail'. **Droite :** La relation conjoint entre les individus de ce contexte.

L'application des méthodes classiques de l'AFC au contexte humain permet de construire une structure conceptuelle incluant des abstractions à partir des descriptions de base des individus, à savoir l'âge et l'expérience de travail. Toutefois, la prise en compte de la relation conjoint par les mécanismes de l'ARC dans la construction d'une telle structure servira sans doute à élargir le champs de découverte de généralisations potentielles à partir des données brutes du contexte humain. Par exemple, on pourrait aboutir à un concept dont la description est la suivante : 'avoir un âge de 27 ans ou moins et être marié à un individu ayant une expérience de travail d'un maximum de 5 années'. Cette description combine clairement des propriétés de base (âge et e.t.) et des liens inter-individus et correspond aux objets o_1 et o_4 du contexte humain.

L'exécution de l'algorithme 3 avec la FCR composée du contexte $\mathcal{K}_h$ et de la relation conjoint de la Table 5.2 s'arrête au bout de deux itérations :

- **Étape d'initialisation :** À l'étape d'initialisation, l'algorithme calcule le contexte dérivé $\mathcal{K}_h^0$ (Figure 5.1, à gauche) à partir de $\mathcal{K}_h$ (Table 5.2) et construit le treillis initial $\mathcal{L}_h^0$ correspondant illustré par la Figure 5.1, à droite. Le vecteur des contextes $\mathbf{K}^+$ et celui des treillis à mettre à jour $\mathbf{L}^+$ contiennent un seul élément, à savoir $\mathcal{K}_h^0$ et $\mathcal{L}_h^0$, respectivement. La structure $\Delta\mathbf{L}^+[1]$ est initialisée à l'ensemble $\{\{c_{\#i} | i = 0 \ldots 9\}$ des concepts du treillis $\mathcal{L}_h^0$. De plus,

`Off[1]=faux.`

	(a) age≤27	(b) age>27	(c) e.t.≤5	(d) e.t.>5
o_1	X		X	
o_2		X	X	
o_3	X		X	
o_4	X			X
o_5		X	X	
o_6		X		X

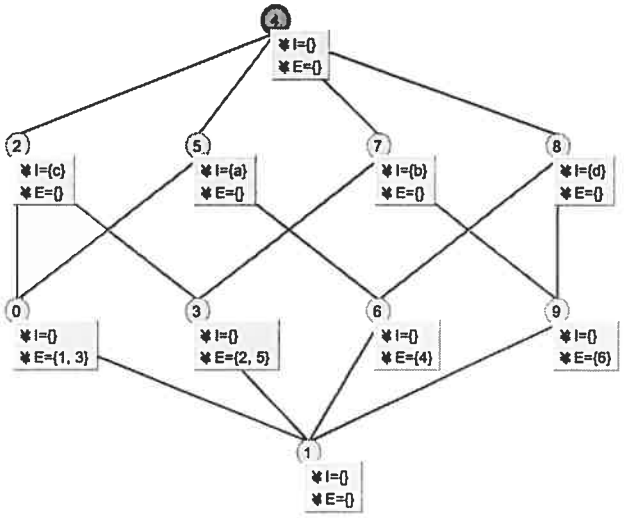


FIG. 5.1 – **Gauche** : Le contexte dérivé $\mathcal{K}_h^0$. **Droite** : Le treillis initial $\mathcal{L}_h^0$.

- **Première itération :** D'abord le treillis actuel du contexte $\mathcal{K}_h$ est déterminé ($\mathbf{L}[1]$=$\mathbf{L}^+[1]$). Les concepts que le mécanisme de scaling utilisera sont aussi déterminés ($\Delta\mathbf{L}[1]$=$\Delta\mathbf{L}^+[1]$). Ensuite, un premier tour de scaling de l'unique relation conjoint est effectué en utilisant $\Delta\mathbf{L}[1]$ qui, rappelons le, contient tous les concepts du treillis initial $\mathcal{L}_h^0$. L'extension relationnelle du contexte $\mathcal{K}_h$ est donnée par la Table 5.3. Le contexte étendu $\mathcal{K}_h^1$ est calculé par la fusion de cette extension avec $\mathcal{K}_h^0$. L'algorithme procède ensuite à la mise à jour du treillis $\mathbf{L}^+[1]$ qui va désormais représenter $\mathcal{L}_h^1$ et est illustré par la Figure 5.2. Afin de connaître si l'on est arrivé au point fixe, l'algorithme calcule $\Delta\mathbf{L}^+[1]$ qui représente les nouveaux concepts dans le treillis $\mathcal{L}_h^1$ par rapport à $\mathcal{L}_h^0$. Ainsi, $\Delta\mathbf{L}^+[1] = \{\{c_{\#i} | i = 10 \ldots 26\}$. Comme les treillis $\mathbf{L}^+[1]$ et $\mathbf{L}[1]$ ne sont pas isomorphes ($\Delta\mathbf{L}^+[1] \neq \emptyset$) le contexte $\mathcal{K}_h$ n'est pas encore saturé et une nouvelle itération devient par conséquent nécessaire.
- **Seconde itération :** Le treillis du contexte $\mathcal{K}_h$ obtenu au bout de la première itération ($\mathbf{L}^+[1]$) devient le treillis actuel ($\mathbf{L}[1]$=$\mathcal{L}_h^1$). Les concepts que le mécanisme de scaling utilisera cette fois-ci sont dans la structure $\Delta\mathbf{L}^+[1]$

	$spouse : c\#0$	$spouse : c\#2$	$spouse : c\#3$	$spouse : c\#4$	$spouse : c\#5$	$spouse : c\#6$	$spouse : c\#7$	$spouse : c\#8$	$spouse : c\#9$
o_1		X	X	X			X		
o_2	X	X		X	X				
o_3				X	X	X		X	
o_4	X	X		X	X				
o_5				X			X	X	X
o_6		X	X	X			X		

TAB. 5.3 – Extension relationnelle du contexte $\mathcal{K}_h$ suivant la relation conjoint au niveau de la première itération.

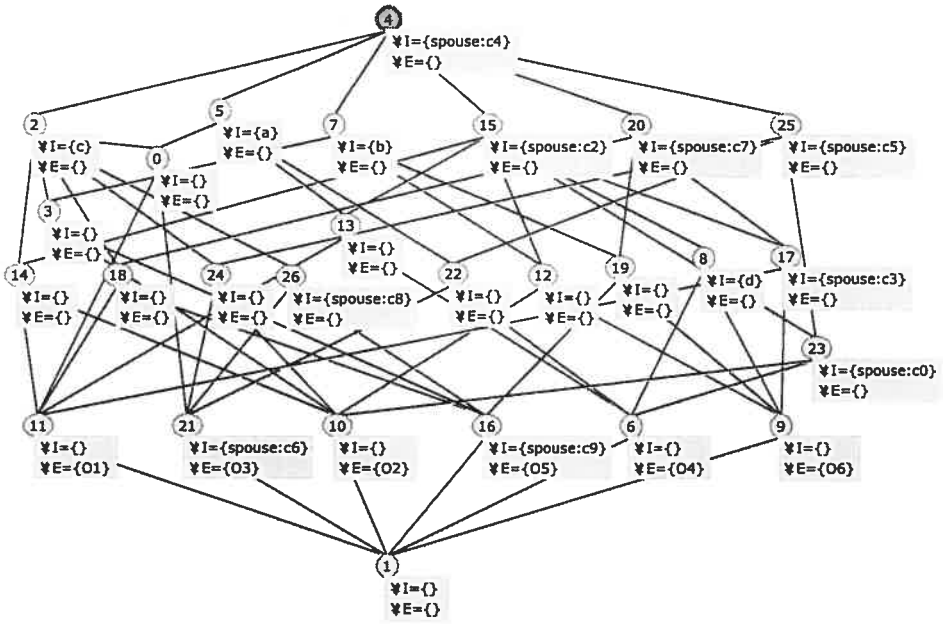


FIG. 5.2 – Le treillis $\mathcal{L}_h^1$ du contexte $\mathcal{K}_h$.

$\{\{c_{\#i}|i = 10 \ldots 26\}$. Le scaling de la relation conjoint donne l'extension relationnelle présentée par la Table 5.4. Le contexte étendu $\mathcal{K}_h^2$ est calculé par la fusion de cette extension avec $\mathcal{K}_h^1$. L'algorithme procède ensuite à la mise à jour du treillis $\mathbf{L}^+[2]$ qui va désormais être $\mathcal{L}_h^2$ et est illustré par la Figure 5.3. L'algorithme calcule ensuite l'ensemble $\Delta\mathbf{L}^+[1]$ qui est s'avère être vide. Ceci signifie que le contexte $\mathcal{K}_h$ est arrivé à un point de saturation et que le processus itératif doit être arrêté. L'algorithme retourne le vecteur des treillis finaux $\mathbf{L}^+$ qui contient le treillis $\mathcal{L}_h^2$ illustré par la Figure 5.3. Nous tenons à préciser que les deux treillis $\mathcal{L}_h^1$ et $\mathcal{L}_h^2$ ne sont pas isomorphes comme treillis de Galois car les intensions de certains concepts sont différentes mais comme des treillis généraux sans étiquetage des nœuds.

	spouse : $c_{\#10}$	*spouse* : $c_{\#12}$	*spouse* : $c_{\#14}$	*spouse* : $c_{\#15}$	*spouse* : $c_{\#23}$	*spouse* : $c_{\#24}$	*spouse* : $c_{\#25}$	*spouse* : $c_{\#11}$	*spouse* : $c_{\#13}$	*spouse* : $c_{\#17}$	*spouse* : $c_{\#18}$	*spouse* : $c_{\#20}$	*spouse* : $c_{\#22}$	*spouse* : $c_{\#21}$	*spouse* : $c_{\#26}$	*spouse* : $c_{\#19}$	*spouse* : $c_{\#16}$
o_1	X	X	X	X	X	X	X										
o_2			X	X				X	X	X	X	X					
o_3				X	X		X		X				X				
o_4						X	X						X	X	X		
o_5		X		X						X		X				X	
o_6											X	X			X	X	X

TAB. 5.4 – Extension relationnelle du contexte $\mathcal{K}_h$ suivant la relation conjoint au niveau de la deuxième itération.

Sur la Figure 5.3, les intensions des concepts contiennent, à la fois, des attributs formels de base et des attributs relationnels induits par le scaling de la relation conjoint. Afin de mettre l'accent sur l'intérêt de la méthode proposée, considérons la paire de concepts mutuellement reliés $c_{\#24} = (\{2, 3\}, \{c, \textit{spouse} : 13\})$ et $c_{\#13} = (\{1, 4\}, \{c, \textit{spouse} : 24\})$ de la Figure 5.3. La Figure 5.4 montre la traduction des attributs formels de ces deux concepts en description initiales des individus du contexte humain. La relation circulaire entre les deux concepts $c_{\#13}$ et $c_{\#24}$ est interprétée par l'existence d'une 'corrélation' entre les personnes dont l'âge est inférieur ou égal à 27 ans et celles ayant un

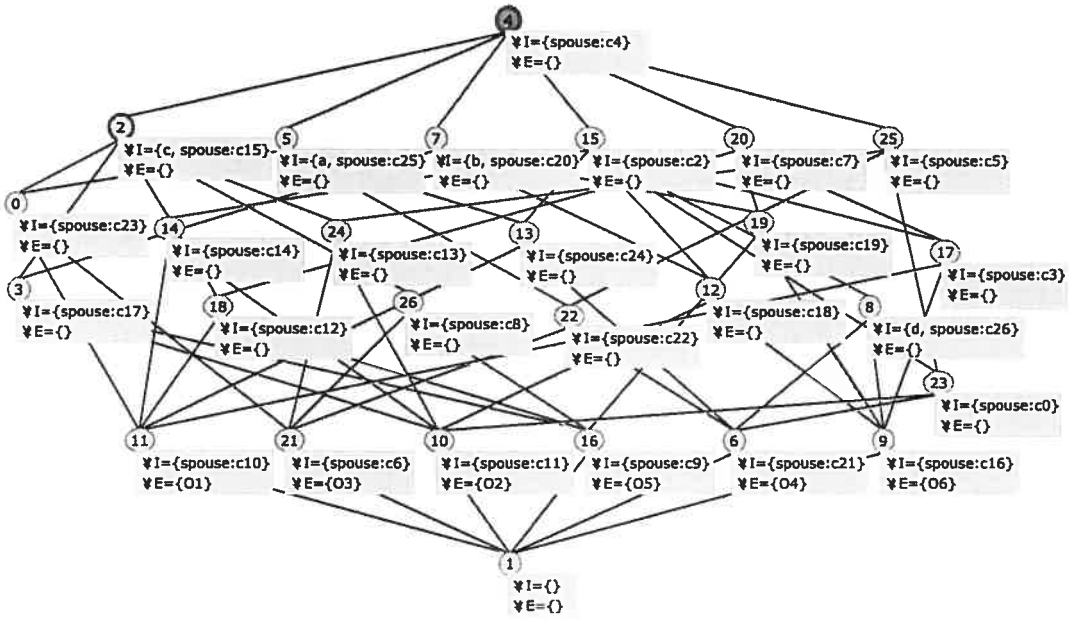


FIG. 5.3 – Le treillis final $\mathcal{L}_h^2$ du contexte $\mathcal{K}_h$.

conjoint dont l'expérience de travail est inférieure à 5 années. De manière inverse, les gens ayant une expérience de travail inférieure ou égale à 5 années sont corrélés à ceux ayant un conjoint dont l'âge est inférieur ou égal à 27 ans. Bien que cet exemple a peu d'intérêt en pratique, la possibilité de trouver une telle information dans des ensembles de données de grande taille peut être très avantageuse (voir Chapitre 8.2.2.2, Section 6.5).

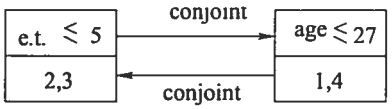


FIG. 5.4 – Exemple de relation inter-concepts circulaire ($c_{\#13}$ et $c_{\#24}$ de la Figure 5.3).

La méthode itérative MULTI-FCA complète les contextes $\mathcal{K}_i$ de la FCR à chaque nouvelle itération par l'extension relationnelle $\Delta\mathcal{K}_i$ issue du mécanisme de scaling. Il y a plusieurs façons de construire, au niveau de l'itération (p+1), les treillis relationnels $\mathcal{L}_i^{p+1}$ (le vecteur $\mathbf{L}^+$ dans l'algorithme 3) à partir de $\mathcal{K}_i^p$

(vecteur $\mathbf{K}[i]$) et de l'extension relationnelle $\Delta\mathcal{K}_i^p$ (vecteur $\Delta\mathbf{K}[i]$). En effet, trois alternatives se présentent :

- **Construction par lot :** construire le treillis complet avec des algorithmes 'par lot' (ou incrémentaux mais à partir de zéro),
- **Construction incrémentale :** obtenir le treillis $\mathcal{L}_i^{p+1}$ avec une mise à jour incrémentale par ajout d'attribut [56] du treillis $\mathcal{L}_i^p$ en utilisant la méthode ADD-ATTRIBUTE (voir plus loin) et en considérant les attributs de $\Delta\mathcal{K}_i^p$ un par un.
- **Fusion de treillis :** obtenir le treillis $\mathcal{L}_i^{p+1}$ par un assemblage de deux treillis [87, 91], c'est à dire $\mathcal{L}_i^{p+1}=\mathcal{L}_i^p \times \Delta\mathcal{L}_i^p$, d'une part le treillis de l'étape précédente $\mathcal{L}_i^p$ et d'autre part, le treillis $\Delta\mathcal{L}_i^p$ de $\Delta\mathcal{K}_i^p$ (extension relationnelle du contexte $\mathcal{K}_i^p$ à l'étape p). À noter que le contexte étendu $\mathcal{K}_i^{p+1}$ est le contexte obtenu par apposition des deux contextes $\mathcal{K}_i^p$ et $\Delta\mathcal{K}_i^p$.

Nous avons retenu la construction incrémentale car elle nous semblait la plus efficace. Chose qui a été confirmée par la suite grâce aux expérimentations que nous avons menées. La section suivante présente le cadre théorique et algorithmique de l'incrémentalité des treillis par ajout d'attribut et donne les algorithmes correspondants.

5.3 Incrémentalité par attribut des treillis/icebergs

D'abord, il faut souligner que le problème de l'incrémentalité des treillis est mis dans un cadre plus général qui est celui de l'incrémentalité des icebergs. Ceci dans le but de réutiliser les résultats des travaux [57] dans le carde du calcul/maintenance des bases de règles d'associations avec les méthodes de l'AFC. Un treillis est un iceberg avec un facteur de coupe $\alpha = 0$ (voir Chapitre 2, Section 2.1.2.7.1). Dans ce qui suit, nous allons présenter les résultats de nos travaux sur l'incrémentalité des treillis/icebergs par ajout d'attributs [56] . L'incrémentalité par ajout d'objets a aussi retenu notre attention dans le cadre d'un autre travail [68, 69].

5.3.1 Incrémentalité par attribut d'un treillis

La construction incrémentale des treillis de concepts par ajout d'attribut est une technique de construction graduelle du treillis qui est très avantageuse pour de nombreuses applications telles que la maintenance des bases de règles d'associations [57], l'exploration et la visualisation des données de grandes taille par le biais de treillis, etc. Une nouvelle perspective d'utilisation de ce type d'incrémentalité s'est présentée dans le cadre de l'ARC. En effet, la méthode MULTI-FCA repose sur l'enrichissement de contextes par de nouveaux attributs formel issus du scaling des relations inter-contextes. Ainsi, il est très utile d'avoir des algorithmes permettant la mise à jour des structures conceptuelles déjà construites (treillis, SHG, etc.) au lieu de procéder à une reconstruction complète du treillis du contexte enrichi.

Dans ce qui suit, nous présentons les fondements théoriques ainsi que les algorithmes mettant en vigueur cette approche de construction et/ou maintenance des treillis de concepts.

5.3.1.1 Fondements théoriques

Soit $\mathcal{K} = (O, A, I)$ un contexte et $\mathcal{L}$ le treillis correspondant. Le contexte $\mathcal{K}^{-1} = (A, O, I^{-1})$ est appelé le contexte "*dual*". En permutant le rôle des objets avec celui des attributs, on obtient aussi un treillis dual. Le principe de la dualité (propriété 2) stipule que la mise à jour d'un treillis par ajout d'attribut peut être réalisé par les opérations duales à celles effectuées dans le cas de la mise à jour par ajout d'objet.

Ainsi, en appliquant le principe de la dualité, l'intégration d'un nouvel attribut a à un treillis $\mathcal{L} = \langle \mathcal{C}, \leq \rangle$ consiste à calculer les intersections de a' avec les extensions des concepts de $\mathcal{C}$ et de trouver les trois catégories de concepts, à savoir, inchangés, modifiés et géniteurs (voir Section 2.2.1 pour la définition de ces catégories). Pour cette raison et à l'instar de l'application $\mathcal{Q}$ qui dans le cadre de l'incrémentalité objet lie les éléments de $\mathcal{C}$ à l'ensemble des sous-ensembles de A ($\mathcal{Q}(X, Y) = Y \cap \{o\}'$), nous définissons l'application $\mathcal{R}$ qui lie les éléments de $\mathcal{C}$ à l'ensemble des sous-ensembles de O de la manière suivante :

Définition 48. $\mathcal{R} : \mathcal{C} \rightarrow 2^O, \mathcal{R}(X,Y) = X \cap \{a\}'$, *où a est l'attribut à ajouter au treillis.*

La fonction $\mathcal{R}$ induit une relation d'équivalence sur $\mathcal{C}$. La classe d'équivalence d'un concept c, notée $[c]_R$, est donnée par :

$$[c]_{\mathcal{R}} = \{\bar{c} \in \mathcal{C} | \mathcal{R}(c) = \mathcal{R}(\bar{c})\}$$

.

Les modifiés $\mathbf{M}(a)$ et les géniteurs $\mathbf{G}(a)$ correspondent aux concepts minimaux[1] de leurs classes d'équivalence dans le treillis $\mathcal{L}$.

Afin que toutes les propriétés qui constituent le cadre théorique de l'incrémentalité par objets [86] puissent être traduites dans le contexte de l'incrémentalité attribut par le principe de la dualité, il est nécessaire de redéfinir les applications σ, γ et χ^+ permettant de lier le treillis $\mathcal{L}$ à $\mathcal{L}^+$. De plus, pour faire la différence entre les deux définitions données pour chacune de ces trois fonctions, on notera σ_o, γ_o et χ_o^+ pour faire référence à la définition dans le cadre de l'incrémentalité objet et σ_a, γ_a et χ_a^+ pour référencer celui de l'incrémentalité attribut.

Les fonctions de correspondance entre $\mathcal{L}$ et $\mathcal{L}^+$ sont définies de la manière suivante :

Définition 49 (σ_a, γ_a et χ_a^+). *Nous définissons les fonctions suivantes :*

- σ_a *: envoie un concept de* $\mathcal{L}$ *vers le concept dans* $\mathcal{L}^+$ *ayant la même extension :*
  $$\sigma_a : \mathcal{C} \rightarrow \mathcal{C},\ \sigma_A(X,Y) = (X,X') \text{ avec } ' \text{ calculer dans } \mathcal{K}^+.$$
- γ_a *: envoie un concept de* $\mathcal{L}^+$ *vers son homologue dans* $\mathcal{L}$ *ayant la même intension modulo a :*
  $$\gamma_a : \mathcal{C}^+ \rightarrow \mathcal{C},\ \gamma_a(X,Y) = (\bar{Y}',\bar{Y}) \text{ où } \bar{Y} = Y - \{a\} \text{ avec } ' \text{ calculé dans } \mathcal{K}.$$
- χ^+ *: retourne pour un concept donné c appartenant à* $\mathcal{L}$ *l'élément minimal de sa classe d'équivalence* $[]_{\mathcal{R}}$ *dans* $\mathcal{L}^+$. $\chi_a^+ : \mathcal{C} \rightarrow \mathcal{C}^+,\ \chi_a^+(X,Y) = (\bar{X},\bar{X}')$ *où* $\bar{X} = X \cap a'$ *avec* $'$ *calculer dans* $\mathcal{K}^+$.

Le pseudo-code de la maintenance du treillis par ajout d'un attribut est donné par l'algorithme 5.

[1] en terme de taille de l'extension

5.3.1.2 Description de la méthode Add-Attribute

La mise à jour du treillis $\mathcal{L}$ suite à l'ajout du nouvel attribut a suit le même schéma que celui de l'ajout d'objet (Chapitre 2, Section 2.2.1), à savoir, calculer les intersections de $\{a\}'$ avec les extensions des concepts de $\mathcal{L}$ ($\mathcal{R}(c)$), déterminer les trois catégories de concepts $\mathbf{M}(a)$, $\mathbf{G}(a)$ et $\mathbf{U}(a)$, effectuer les modifications nécessaires et enfin mettre à jour la relation de couverture. L'algorithme 5 présente le pseudo-code de ces opérations. Il a été obtenu à partir de l'algorithme ADD-OBJECT (Algorithme 1) par principe de dualité.

L'algorithme 5 commence par trier les concepts par ordre décroissant des extensions (ligne 3) afin de permettre un parcourt descendant du treillis. Ensuite, l'algorithme parcourt l'ensemble des concepts (ligne 4). Pour chaque concept c, on calcule les intersections générées par les prédécesseurs du concept c avec l'extension du nouvel attribut et on leur applique la fonction ARGMAX qui retourne le prédécesseur ayant produit la plus grande intersection. Si l'image du prédécesseur produisant la plus grande valeur de $\mathcal{R}$ est égale à celle du concept c par la même application $\mathcal{R}$ alors le concept courant c n'est pas maximal dans sa classe d'équivalence et est un concept inchangé ($c \in \mathbf{U}(a)$). Par contre, si les deux images sont différentes (ligne 6), alors le concept c est l'élément maximal de sa classe. À ce moment, son statut doit être déterminé :

- Si la taille de l'ensemble $\mathcal{R}(c)$ est égale à celle de l'extension de c (ligne 7), alors c est un concept modifié car $Ext(c) \subseteq a'$. L'algorithme modifie l'intension de c (ligne 8), le concept c est ajouté à l'ensemble des modifiés $\mathbf{M}(a)$ (ligne 9) et l'élément minimal de la classe d'équivalence est mis à jour (ligne 10). Ces deux dernières tâches sont essentielles pour la mise à jour de la relation d'ordre dans le nouveau treillis.
- Sinon, c est un concept géniteur. Par conséquent, un nouveau concept $\hat{c}$ est crée dont l'extension et l'intension sont $\mathcal{R}(c)$ et l'intension de c à laquelle on rajoute le nouvel attribut a, respectivement (ligne 12). Ensuite, on calcule un ensemble de prédécesseurs candidats composé des images dans $\mathcal{L}^+$

```
1: procedure ADD-ATTRIBUTE(In/Out : L = ⟨C, ≤⟩ a lattice; In : a an attribute)
2: SORT(C)
3: for all c in C do
4:    new-min ← ARGMAX({|R(c̄)| | c̄ ∈ Cov^l(c)})
5:    if |R(c)| ≠ |R(new-min)| then
6:       if |R(c)| = |Ext(c)| then
7:          Int(c) ← Int(c) ∪ {a}      {c is modified}
8:          M(a) ← M(a) ∪ {c}
9:          new-min ← c
10:      else
11:         ĉ ← NEW-CONCEPT(R(c), Int(c) ∪ {a})      {c is genitor}
12:         Candidates ← {ChiPlus(c̄) | c̄ ∈ Cov^l(c)}
13:         for all c̄ in MIN-CONCEPTS(Candidates) do
14:            NEW-LINK(ĉ, c̄)
15:            if c̄ ∈ M(a) then
16:               DROP-LINK(c, c̄)
17:         new-min ← ĉ
18:         L ← L ∪ {ĉ}
19:   ChiPlus(c) ← new-min
```

Algorithm 5: Mise à jour du treillis suite à l'ajout d'un nouvel attribut au contexte.

des concepts de la couverture inférieure de c par l'application χ_a^+ (ligne 13). Puis, on génère un lien entre le nouveau concept $\hat{c}$ et les concepts minimaux retournés par la primitive MIN-CONCEPTS qui filtre les candidats non comparables et ayant la plus petite intension (ligne 15). On regarde aussi parmi les concepts minimaux $\overline{(c)}$ ceux qui sont modifiés (ligne 16) afin de supprimer leurs liens avec le concept courant c (ligne 17). Le minima de la classe d'équivalence est positionné au nouveau concept $\hat{c}$ (ligne 18) qui est aussi rajouté à la structure $\mathcal{L}^+$ (ligne 19).

Finalement, la structure représentant l'application χ_a^+ est mise à jour par l'association de l'image du concept c par χ_a^+ au concept minimal de la classe d'équivalence de c.

Exemple :

Soient le treillis $\mathcal{L}$ à gauche de la Figure 5.5 et un nouvel attribut h avec $h' = \{1, 3, 5\}$. La Table 5.5 illustre pour chaque concept c de $\mathcal{L}$, la valeur de $\mathcal{R}(c)$, son statut et le nouveau concept minimal de la classe d'équivalence $[c]_{\mathcal{R}}$.

c	$c_{\#1}$	$c_{\#4}$	$c_{\#8}$	$c_{\#7}$	$c_{\#5}$	$c_{\#0}$	$c_{\#3}$	$c_{\#6}$	$c_{\#2}$
$\mathcal{R}(c)$	135	1	15	35	15	$\emptyset$	1	5	$\emptyset$
$\chi^+(c)$	$c_{\#1}$	$c_{\#3}$	$c_{\#5}$	$c_{\#7}$	$c_{\#5}$	$c_{\#2}$	$c_{\#3}$	$c_{\#6}$	$c_{\#2}$
Status	*gen*	*inc*	*inc*	*mod*	*mod*	*inc*	*mod*	*mod*	*mod*

TAB. 5.5 – Trace d'exécution de l'algorithme 5 avec le treillis sur la gauche de la Figure 5.5.

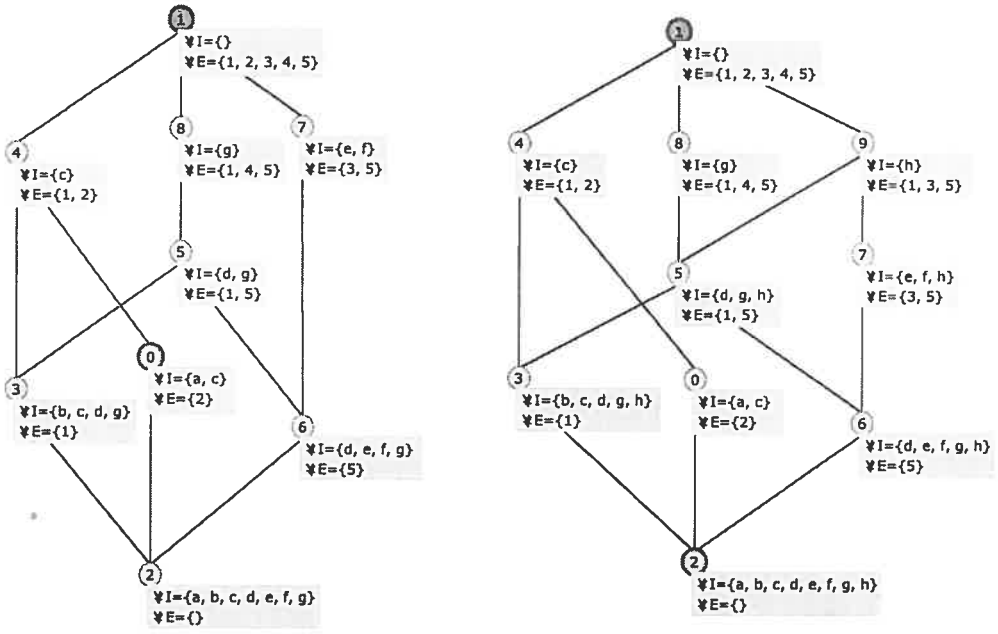


FIG. 5.5 – **Gauche :** le treillis $\mathcal{L}$. **Droite :** le treillis $\mathcal{L}^+$ obtenu après l'ajout de $(135, h)$.

La Figure 5.5 montre le treillis $\mathcal{L}^+$. L'ensemble des concepts par catégories sont : $\mathbf{U}^+(h) = \{c_{\#0}, c_{\#4}, c_{\#8}\}$; $\mathbf{M}^+(h) = \{c_{\#2}, c_{\#3}, c_{\#5}, c_{\#6}, c_{\#7}\}$; $\mathbf{G}^+(h) = \{c_{\#1}\}$ et $\mathbf{N}^+(h) = \{c_{\#9}\}$.

L'exécution de l'algorithme 5 est comme suit : d'abord le tri des concepts dans l'ordre décroissant de la taille des extensions donne $\{c_{\#1}, c_{\#4}, c_{\#8}, c_{\#7}, c_{\#5}, c_{\#0}, c_{\#3}, c_{\#6}, c_{\#2}\}$. Le traitement du premier concept $c_{\#1}$ donne lieu au nouveau concept $c_{\#9} = (135, h)$. En effet, $c_{\#1}$ et l'unique concept dans sa classe d'équivalence et par conséquent il est aussi le concept minimal. Comme $|\mathcal{R}(c_{\#1})| \neq |Ext(c_{\#1})|$, alors $c_{\#1}$ est un géniteur et le nouveau concept $c_{\#9} = (\mathcal{R}(c_{\#1}), Int(c_{\#1}) \cup \{h\}) = (135, h)$. La mise à jour de la relation d'ordre dans le treillis commence par la détermination des prédécesseurs du concept $c_{\#1}$ dans le treillis actuel qui sont $\{c_{\#4}, c_{\#8}$ et $c_{\#7}\}$; ensuite des concepts minimaux dans leurs classes d'équivalence (χ^+), à savoir, $\{c_{\#3}, c_{\#5}$ et $c_{\#7}\}$, respectivement. Les concepts minimaux en taille d'intension de $\{\{c_{\#3}, c_{\#5}, c_{\#7}\}$ sont $\{c_{\#5}, c_{\#7}\}\}$ (ligne 14). Par conséquent, deux liens sont créés entre le nouveau concept $c_{\#9}$ et les deux concepts $c_{\#5}$ et $c_{\#7}$ (ligne 15). Le lien entre le concept $c_{\#1}$ et $c_{\#7}$ est supprimé car $c_{\#7}$ est un modifié(ligne 17). Le traitement des autres concepts donne lieu à des concepts modifiés ou inchangé comme le montre la Table 5.5.

5.3.1.3 Complexité de Add-Attribute

Soient $l = \|\mathcal{C}_\mathcal{K}^+\|$, $m = \|A\|$, $k = \|O\|$ et $\Delta l = \|\mathcal{C}_\mathcal{K}^+\| - \|\mathcal{C}_\mathcal{K}\|$. Le tri des concepts est linéaire dans la taille du treillis car en effet il s'agit d'une comparaison entre les tailles des intensions qui sont bornées par m. Le calcul de $\mathcal{R}(c)$ est borné par n, car il s'agit d'intersection d'extensions. La couverture inférieure d'un concept $c = (X, Y)$ contient un nombre de concepts qui est borné par $m - \|Y\|$, en effet un concept $\bar{c} = (\bar{X}, \bar{Y})$ est un prédécesseur de c si $\bar{Y} = Y \cup \bar{A}$ tel que $\bar{A} \subseteq A \setminus Y$ et ainsi le calcul de la complexité de l'opération ARGMAX appartient à $O(n + m)$. La mise à jour de l'ordre, effectuée pour chaque géniteur, a un coût de $O(m^2)$ qui peut s'expliquer par le parcours de la couverture inférieure ($O(m)$) et pour chaque élément de la couverture, un calcul d'intersection sur les intensions également en $O(m)$. Les autres

opérations étant négligeables à celle là (elles sont à temps constant). Donc, la partie liée aux géniteurs a un coût global en $O(\Delta l \cdot n^2)$. Le coût total de l'ajout d'un attribut est en $O(l \cdot (n+m) + \Delta l \cdot m^2)$. En fin, la complexité de l'algorithme pour l'ajout de tous les attributs est de l'ordre de $O(ml \cdot (m+n))$.

5.3.2 Incrémentalité par attribut d'un iceberg

Dans le Chapitre 2, section 2.1.2.7, nous avons défini l'iceberg comme étant des ensembles supérieurs obtenus par une restriction sur la cardinalité de l'extension des concepts du treillis complet.

Les opérations de mise à jour d'un iceberg suite à l'ajout d'un nouvel attribut préservent les concepts existants et pouvant créer de nouveaux concepts dont la fréquence satisfait le seuil α établi. Nous allons, d'abord, énoncer les deux propriétés qui régissent ces deux types de concepts dans l'iceberg ; ensuite, décrire l'algorithme de mise à jour proposé.

Exemple : La Figure 5.7 montre l'iceberg $\bar{\mathcal{L}}^{0.2+}$ obtenu par ajout de (135,h) à l'iceberg $\bar{\mathcal{L}}^{0.2}$ de la Figure 5.6.

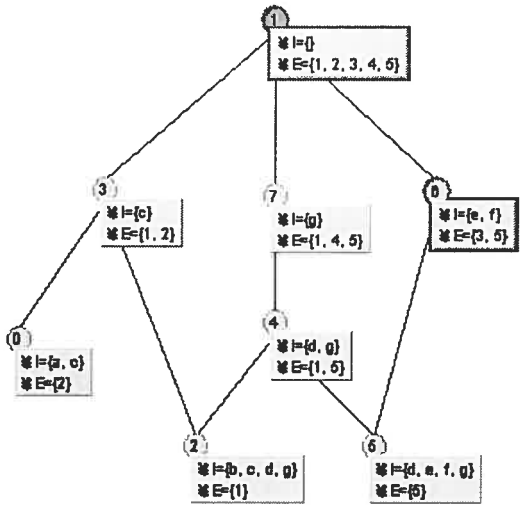


FIG. 5.6 – L'iceberg $\bar{\mathcal{L}}^{0.2}$.

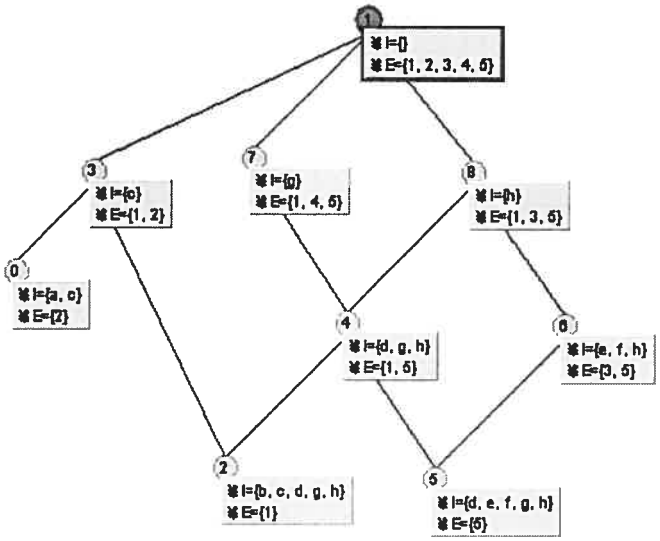


FIG. 5.7 – L'iceberg $\tilde{\mathcal{L}}^{0.2+}$ obtenu par ajout de (135,$\{h\}$) à l'iceberg $\tilde{\mathcal{L}}^{0.2}$ de la Figure 5.6.

5.3.2.1 Fondements théoriques

L'iceberg est composé des concepts fréquents du treillis complet. L'ajout d'un attribut n'affecte pas la fréquence (application ν) des concepts présents car cette dernière dépend uniquement de l'extension du concept et de l'ensemble des objets ($\nu(c) = \|X\|/\|O\|$). Ainsi, une fois qu'un concept est créé dans l'iceberg, son statut de fréquent demeure inchangé avec l'évolution de l'ensemble des attributs.

Propriété 23. *Soient $\tilde{\mathcal{L}}^{\alpha} = \langle \tilde{C}^{\alpha}, \leq \rangle$ et $\tilde{\mathcal{L}}^{\alpha+} = \langle \tilde{C}^{\alpha+}, \leq_{+} \rangle$ les icebergs obtenus à partir des deux contextes $\mathcal{K} = (O, A, I)$ et $\mathcal{K}^{+} = (O, A^{+}, I^{+})$, respectivement, avec $A^{+} = A \cup \{a\}$ et $I^{+} = I \cup (a' \times \{a\})$. Nous avons : $\tilde{C}^{\alpha} \subseteq \tilde{C}^{\alpha+}$.*

Preuve. Soit c un concept de $\tilde{\mathcal{L}}^{\alpha}$. La fréquence de c, $\nu(\sigma_a(c))$ dans $\tilde{\mathcal{L}}^{\alpha+}$ est égale à $\frac{\|Ext(\sigma_a(c))\|}{\|O\|}$. Comme $Ext(\sigma_a(c)) = Ext(c)$ et que le nombre d'objets ne varie pas lors du passage de $\tilde{\mathcal{L}}^{\alpha}$ à $\tilde{\mathcal{L}}^{\alpha+}$, on a alors $\nu(c) = \nu(\sigma_a(c))$. ■

D'autre part, les nouveaux concepts (dits 'jumpers') représentent les bornes supérieures fréquentes obtenues lors du calcul des intersections $\mathcal{R}(c)$ ($\mathcal{R}(c) =$

$Ext(c) \cap \{a\}'$, (Définition 48) entre les concepts de l'iceberg et le concept-attribut $\mu(a)$. Un parcours descendant (top-down) de l'iceberg permet de filtrer les nouveaux concepts fréquents et d'éviter de calculer les prédécesseurs d'un concept non fréquent car ils sont aussi non fréquents comme indiqué par la propriété suivante :

Propriété 24. $\forall c, \underline{c} \in \bar{\mathcal{L}}^{\alpha}$ *avec* $\underline{c} \prec c$, *si* $\|\mathcal{R}(c)\| \leq \alpha\|O\|$ *alors* $\|\mathcal{R}(\underline{c})\| \leq \alpha\|O\|$.

Preuve. : En effet, $\underline{c} \prec c \Rightarrow \underline{X} \subset X \Rightarrow (\underline{X} \cap a') \subset (X \cap a') \Rightarrow \|\underline{X} \cap a'\| \leq \|X \cap a'\| \leq \alpha$. ■

5.3.2.2 La méthode Magalice-A

Les opérations de mise à jour correspondantes suite à l'ajout d'un nouvel attribut ressemblent de très près à celles d'un treillis complet ADD-ATTRIBUTE (Algorithme 5). En effet, Nous avons introduit le filtrage des concepts fréquents à la méthode ADD-ATTRIBUTE pour aboutir au pseudo-code suivant permettant de mettre à jour un iceberg suite à l'ajout d'un nouvel attribut.

La méthode commence par un tri linéaire des concepts par ordre décroissant de la taille des extensions pour permettre un parcours descendant de l'iceberg (ligne 5). Ensuite, pour chaque concept, on vérifie en premier lieu s'il n'est pas un prédécesseur d'un concept ayant produit une intersection non fréquente (en vertu de la propriété 24). Pour ce faire, on utilise la primitive LOOKUP qui accepte en entrée un concept c ainsi que l'ensemble des concepts non fréquents, appelé `unfrequent`, et retourne le successeur de c (ligne 7). Si un tel successeur n'existe pas, on calcule alors la valeur de $\mathcal{R}(c)$ et on teste sa fréquence (ligne 10). Si elle est fréquente, on applique les opérations effectuées sur le treillis complet (ligne 11-25). Sinon, on ajoute tous les prédécesseurs de c à l'ensemble `unfrequent` (lignes 27-28) afin de limiter le calcul des intersections à celles qui sont, probablement, fréquentes. À la fin, l'algorithme utilise la primitive CLEAN pour éliminer les concepts non fréquents et mettre à jour l'ordre dans l'iceberg.

Exemple : Étant donné l'iceberg $\bar{\mathcal{L}}^{0.2}$ de la Figure 5.6 et le nouvel attribut h avec $h' = \{1, 3, 5\}$. La Table 5.6 montre le filtrage des concepts de $\bar{\mathcal{L}}^{0.2}$ par

```
 1: procedure MAGALICE-A(In/Out : L̄^α = ⟨C̄^α, ≤⟩ an iceberg; In : a an attribute, α a
    minimum support)
 2:   Local   : unfrequent : set
 3:
 4: unfrequent ← ∅
 5: SORT(C̄^α)
 6: for all c in C̄^α do
 7:   ĉ ← LOOKUP(unfrequent, c)
 8:   if (ĉ = NULL) then
 9:     if |R(c)| ≥ α · |O| then
10:       new-min ← ARGMAX({|R(c̄)| | c̄ ∈ Cov^l(c)})
11:       if |R(c)| ≠ |R(new-min)| then
12:         if |R(c)| = |Ext(c)| then
13:           Int(c) ← Int(c) ∪ {a}        {c is modified}
14:           M(a) ← M(a) ∪ {c}
15:           new-min ← c
16:         else
17:           ĉ ← NEW-CONCEPT(R(c),Int(c) ∪ {a})        {c is genitor}
18:           Candidates ← {ChiPlus(c̄) | c̄ ∈ Cov^l(c)}
19:           for all c̄ in MIN-CLOSED(Candidates) do
20:             NEW-LINK(ĉ,c̄)
21:             if c̄ ∈ M(a) then
22:               DROP-LINK(c, c̄)
23:           new-min ← ĉ
24:           C̄^α ← C̄^α ∪ {ĉ}
25:           ChiPlus(c) ← new-min
26:     else
27:       for all c̄ in Cov^l(c) do
28:         unfrequent ← unfrequent ∪ {c̄}
29:   CLEAN(L̄^α,unfrequent)
```

Algorithm 6: Mise à jour de l'iceberg suite à l'ajout d'un nouvel attribut au contexte.

l'Algorithme 6 pour le calcul de l'iceberg $\bar{\mathcal{L}}^{0.2+}$.

c	$\mathcal{R}(c)$	$\lvert\mathcal{R}(c)\rvert \geq 1$	$\chi^{+}(c)$	Statut	Var. `unfrequent`
$\{c_{\#1}\}$	135	OUI	$\{c_{\#1}\}$	géniteur	$\emptyset$
$\{c_{\#3}\}$	1	OUI	$\{c_{\#2}\}$	inchangé	$\emptyset$
$\{c_{\#7}\}$	15	OUI	$\{c_{\#4}\}$	inchangé	$\emptyset$
$\{c_{\#6}\}$	35	OUI	$\{c_{\#6}\}$	modifié	$\emptyset$
$\{c_{\#4}\}$	15	OUI	$\{c_{\#4}\}$	modifié	$\{c_{\#0}\}$
$\{c_{\#0}\}$	$\emptyset$	NON	$\{c_{\#0}\}$	inchangé	$\{c_{\#0}\}$
$\{c_{\#2}\}$	1	OUI	$\{c_{\#2}\}$	modifié	$\{c_{\#0}\}$
$\{c_{\#5}\}$	5	OUI	$\{c_{\#5}\}$	modifié	$\{c_{\#0}\}$

TAB. 5.6 – Trace de l'exécution de la méthode MAGALICE-A avec l'iceberg $\bar{\mathcal{L}}^{0.2}$ lors de l'ajout de l'attribut h.

5.3.2.3 Complexité de Magalice-A

À l'instar de MAGALICE-O, La table ci-dessous montre les paramètres utilisés dans l'estimation de la complexité en pire cas de l'algorithme MAGALICE-A.

Variable	m	l	n
Signification	$\lvert\lvert A\rvert\rvert$	$\lvert\lvert\mathcal{L}^{\alpha+}\rvert\rvert$	$\lvert\lvert O\rvert\rvert$

La complexité de la méthode MAGALICE-A est identique à celle de mise à jour du treillis complet (ADD-ATTRIBUTE). En effet, la fonction LOOKUP (ligne 7) est de l'ordre de n, le test de la fréquence de $\mathcal{R}$ est de l'ordre $O(1)$ (ligne 9) et l'ajout des prédécesseurs d'un concept à l'ensemble des concepts non fréquents est de l'ordre de m. Et par conséquent, en faisant la somme totale des complexités de toutes les fonctions de l'algorithme 6, on obtient la complexité de l'algorithme de la mise à jour du treillis complet qui est $O(ml \cdot (m+n))$ (voir Section 5.3.1.3).

5.4 Conclusion

Comme contribution aux développements algorithmiques de l'approche proposée, nous avons mis au point une adaptation de la méthode de calcul de concepts

relationnels (MULTI-FCA) au treillis complets en utilisant le scaling relationnel. Le principe de l'approche consiste à aboutir à la solution finale (treillis finaux) en calculant un point fixe d'une fonction qui, essentiellement, associe à une collection de treillis (un par contexte), une autre collection de treillis plus raffinée suivant un processus itératif. À chaque itération, la fonction utilise d'abord les concepts découverts à l'étape précédente pour le scaling des liens inter-objets. Ainsi, des échelles pour les différentes liens inter-objets sont créées et permettent d'enrichir les descriptions des objets. À la fin de chaque itération, la fonction construit les treillis qui correspondent aux contextes mis à jours. Nous avons aussi décrit dans ce chapitre nos recherches sur l'incrémentalité par attribut des treillis et icebergs. Les résultats de ces recherches ont été utilisés avec la méthode de l'ARC et dans le cadre d'autres travaux reliés à la maintenance des bases de règles d'association.

Chapitre 6

ARC et réingénierie des modèles UML

Dans ce chapitre, nous allons, d'abord décrire comment coder un modèle de classes UML[1] en une famille de contextes (FCR), le format d'entrée de l'ARC. Nous allons aborder les difficultés que rencontre un tel codage notamment celles qui émergent des conflits de noms des différents éléments du modèle. Par la suite, nous allons décrire de façon générale les règles de traduction des résultats de l'analyse (une famille de treillis relationnels) en modèle UML. Par la même occasion, nous allons illustrer le déroulement du principal algorithme de l'ARC, MULTI-FCA, sur un exemple de petite taille.

6.1 ARC dans la refactorisation des modèle UML

La construction de modèles statiques UML est une activité clé dans le processus de conception du logiciel orienté objet. Dans un tel modèle, la relation de spécialisation/généralisation est le facteur structurant car elle permet la répartition optimale des spécifications sur un ensemble de classes organisées hiérarchiquement et par là sert de support pour le mécanisme d'héritage. Cependant, il n'y a pas

[1]Version 1.5 ou plus.

de méthodologie standard pour la construction d'une hiérarchie de classes à partir d'un ensemble de classes munies de leurs propriétés (variables et méthodes). Tout au plus, un ensembles de critères consensuels pour ce qu'est une hiérarchie "optimale" se trouve parfois de façon implicite dans la littérature [92].

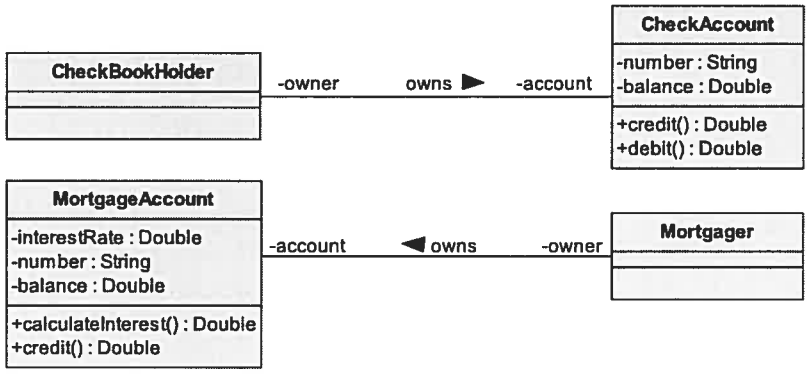


FIG. 6.1 – Les classes du domaine de la gestion des comptes bancaires.

6.1.1 Rappel sur l'AFC et la construction des hiérarchies

Plusieurs techniques basées sur l'analyse de données ou l'apprentissage automatique ont été étudiées en tant qu'outils d'aide à la construction de hiérarchies, chacune introduisant dans le processus de recherche d'une hiérarchie convenable ses propres biais, explicites ou implicites. L'AFC [30, 95] de part 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 [78] et de construction [32, 19, 39] de hiérarchies de classes ainsi que la découverte de patrons de conception [82]. 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 per-

met d'avoir une structure sans redondances et favorise par conséquent l'héritage en orienté objet.

	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

TAB. 6.1 – Le contexte binaire des classes du diagramme UML de la Figure 6.1.

L'AFC a été utilisée avec succès dans la restructuration des hiérarchies de classes de niveau conception et implémentation [19, 32, 34]. Ainsi, l'entrée typique d'un outil de factorisation basé sur FCA (par exemple, Kaba [79]) est un ensemble de classes, éventuellement hiérarchisé, alors que la sortie est une hiérarchie qui est maximalement factorisée, c'est-à-dire que tout élément de spécification n'est localisé qu'à un seul endroit dans la hiérarchie.

Le codage classique d'un modèle de classes en un contexte formel consiste à considérer les classes comme étant des objets formels et leurs propriétés (attributs et/ou méthodes) comme étant des attributs formels. Cette façon de codage permet au processus d'analyse de l'AFC de déterminer tous les groupes de classes ayant des propriétés similaires.

Exemple : la Table 6.1 montre le codage des classes représentant le domaine des comptes bancaires du diagramme de classes de la Figure 6.1. La Figure 6.2 montre le treillis construit à partir de ce codage. Les objets formels `CBH`, `CA`, `S`, `M`, `MA` représentent les abréviations des classes `CheckBookHolder`, `CheckAccount`,

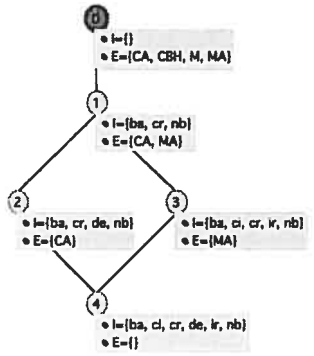


FIG. 6.2 – Le treillis des concepts associé au contexte de la Table 6.1.

`String`, `Mortgager` et `MortgageAccount`, respectivement.

L'AFC permet de découvrir des abstractions potentiellement utiles dans un diagramme de classes. En effet, la Figure 6.3 illustre une interprétation partielle en un diagramme de classes. On constate la présence d'une classe plus abstraite, appelée `Account` qui généralise les deux types de comptes bancaires `CheckAccount` et `MortgageAccount`.

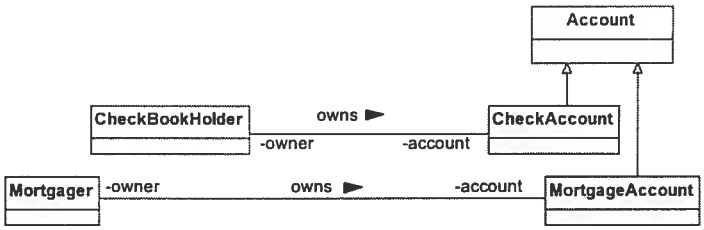


FIG. 6.3 – L'interprétation du treillis de la Figure 6.1 en un diagramme de classes.

6.1.2 L'ARC et la construction des modèles de classes

L'ARC [16] a été, initialement, introduite afin d'étendre la portée de l'approche AFC dans la construction des modèles de classes. En effet, elle permet de construire des abstractions sur tous les types d'éléments du modèle : attributs, rôles, asso-

ciation, etc. Ceci est rendu possible grâce, entre autres, au format d'entrée étendu de l'ARC et de ses mécanismes de formation de concepts qui utilisent les similarités dans les liens inter-individus pour établir des relations inter-concepts. La Figure 6.4 illustre le processus de restructuration d'un modèle UML. Ce processus est composé de trois principales étapes, à savoir : le codage du modèle UML, la dérivation des treillis et enfin l'exploitation de ces treillis pour la génération d'un modèle structuré.

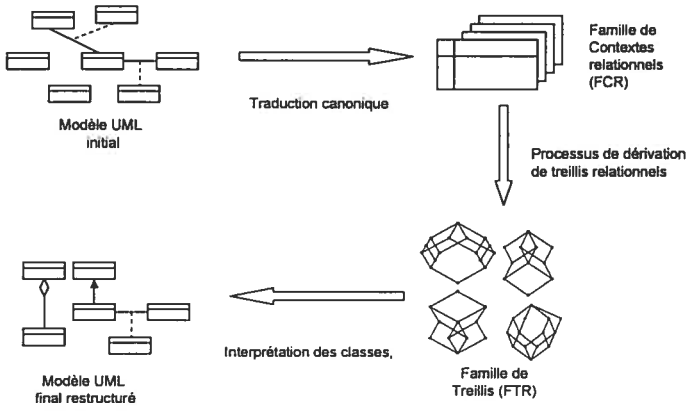


FIG. 6.4 – Processus de restructuration d'un modèle UML au moyen de l'ARC.

Alors que l'AFC admet exclusivement des données sous forme de tables représentant un ensemble d'individus possédant des propriétés binaires tel que montré par la Table 6.1, le format d'entrée de l'ARC admet des relations inter-individus. Cela permet de décomposer un modèle de classes UML en plusieurs contextes, un par type d'éléments du modèle qui sera sujet à un processus d'abstraction (typiquement, classes et associations) et de relier ces contextes par le biais de relations techniques inter-contextes. Ces relations sont à sémantique indépendante du domaine sous-jacent et traduisent les liens qui existent entre éléments au sein du modèle pour leur permettre de former un tout (par exemple, le lien "possède" entre une classe et un attribut). L'avantage d'un tel codage et de la prise en compte ultérieure des informations relationnelles est la possibilité d'abstraire sur plusieurs types d'éléments

tout en opérant des enrichissements mutuels entre hiérarchies d'abstractions sur des types différents reliés par des relations techniques.

	name=owns	originRole=owner	destRole=account	originType=CBH	originType=M	destType=CA	destType=MA	destType=Account
CBH-CA.owns	X	X	X	X		X		X
M-MA.owns	X	X	X		X		X	X

TAB. 6.2 – Le contexte binaire des associations du diagramme de classes de la Figure 6.1.

Exemple : la Table 6.2 montre une manière de coder les associations du modèle UML de la Figure 6.1. Un préfixe a été ajouté aux noms des associations ayant des noms identiques. Ce prefixe est composé des noms des classes reliées. Les associations peuvent être décrites par des propriétés binaires telles que le nom (`name=owns`) et par des relations avec les classes du modèle telles que la relation `originType` et `destType`. Par la suite, nous allons donner un ensemble de propriétés décrivant les associations en s'appuyant sur le méta-modèle UML. Les deux propriétés `originType` et `destType` sont échantillonnées de façon nominale et donnent les propriétés binaires suivantes : `originType=CBH`, `originType=M`, `destType=CA` et `destType=MA`. La classe `Account` découverte lors du calcul des abstractions potentielles dans le contexte des classificateurs (voir Figure 6.2) est intégrée au codage des associations comme le montre la Table 6.2 par le biais de la colonne `destType=Account`. Cela signifie que les instances des classes de destination des deux associations `owns` sont aussi instances de la classe `Account` car les deux classes de destination `CheckAccount` et `MortageAccount` sont des sous-classes de `Account`. Ceci est l'œuvre du mécanisme de scaling relationnel (voir Chapitre 4, Section 4.5.1) qui fait en sorte que les concepts découverts dans un contexte sont intégrés à tous les contextes qui sont reliés à ce contexte comme étant de nouveaux attributs formels.

Le treillis de concepts du contexte des associations donné par la Figure 6.5

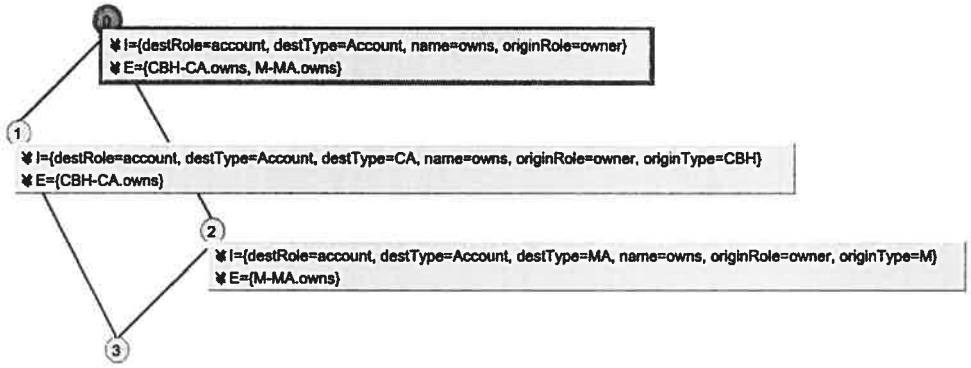


FIG. 6.5 – Treillis du contexte des associations du modèle UML de la Figure 6.1.

contient le concept $c_{\#0}$ dont l'interprétation est une nouvelle association, appelée aussi `owns` qui généralise les deux associations, celle qui lie `Mortgager` à `MortageAccount` et celle qui lie `CheckBookHolder` à `CheckAccount` comme le montre la Figure 6.6.

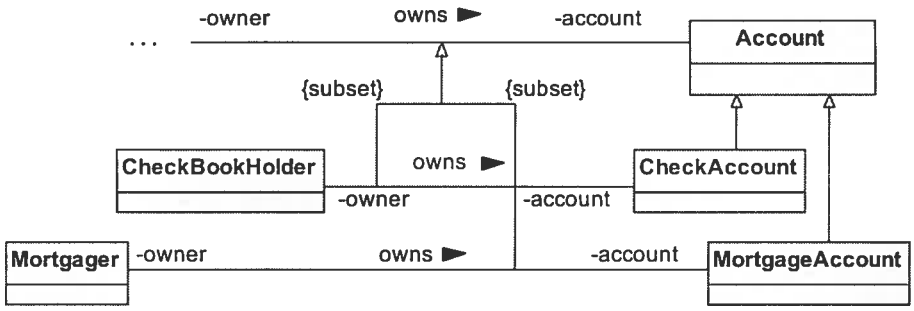


FIG. 6.6 – Découverte d'une association `owns` plus générale en se basant sur le treillis des associations de la Figure 6.5. Le treillis en question ne précise pas l'autre extrémité de cette nouvelle association.

Dans ce qui suit, nous allons présenter un codage du modèle UML permettant de calculer des généralisations sur tous les types d'éléments appartenant à ce modèle. De plus, nous allons nous appuyer sur le méta-modèle UML pour dégager les propriétés intrinsèques de chaque type d'éléments ainsi que les relations entre ces éléments.

6.2 Construction de la FCR d'un modèle de classes

Afin de former la FCR d'un modèle de classes UML nous avons examiné les éléments du méta-modèle UML du diagramme de classes pour dégager ceux qui présentent un intérêt pour la découverte d'abstractions potentiellement utiles. Ainsi, les classes du méta-modèle, les propriétés des classes et les associations sont des contextes, des attributs formels et des relations dans la FCR, respectivement. Cependant, le méta-modèle UML contient beaucoup de détails qui sont inutiles pour le processus d'analyse/restructuration basé sur l'ARC. Par exemple, chaque classe est décrite au niveau du méta-modèle par la propriété booléenne `isLeaf` qui détermine si une classe donnée peut avoir des classes dérivées ou non. Étant données deux classes dont la propriété correspondante `isLeaf` est positionnée à `Vrai`, la méthode de l'ARC créerait une nouvelle classe plus générale dont la propriété `isLeaf` est positionnée à `Vrai` aussi alors que cette dernière classe est une superclasse des deux autres classes.

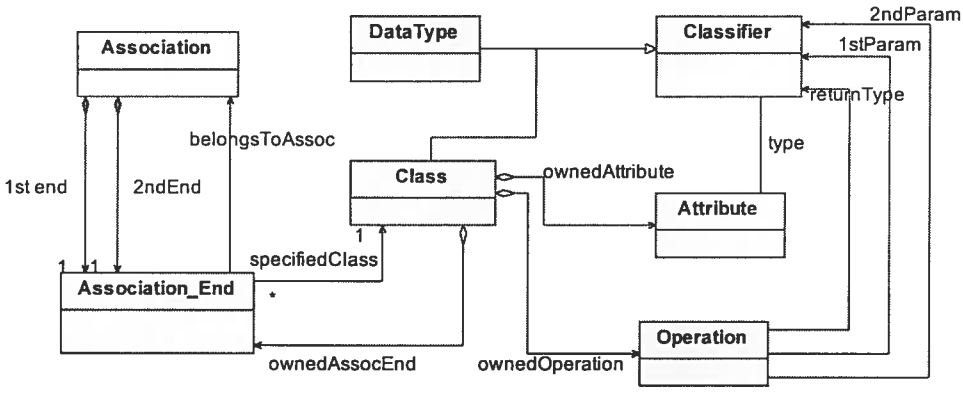


FIG. 6.7 – Le sous-méta-modèle du diagramme de classes utilisé pour la construction de la FCR.

Les contextes retenus dans le cadre de nos expérimentations sont les contextes suivants : `Classifier`, `Association`, `Association_End`, `Attribute` et `Operation` ainsi que les relations qui les relient telles qu'elles sont énoncées par le méta-modèle,

à savoir, `type`, `owned_attribute`, `owned_operation`, `owned_association_end`, `return_type`, `belong_to_association`, `first_end`, `second_end` et `specified_class`. Ci-dessous, nous décrivons le contenu de ces différents contextes.

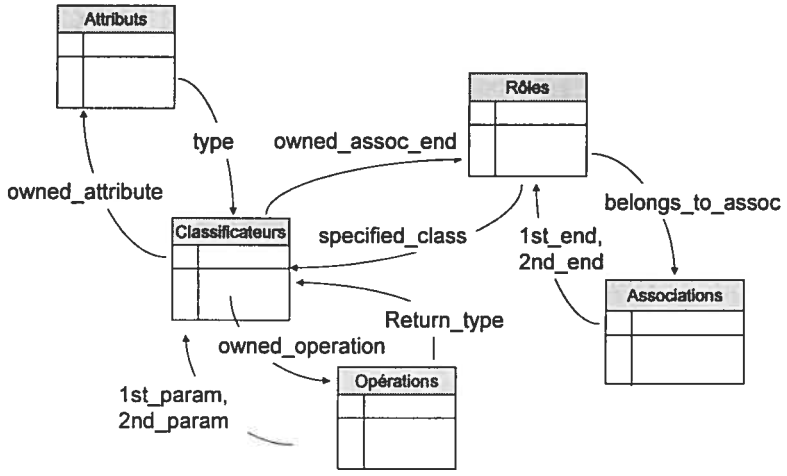


FIG. 6.8 – FCR d'un diagramme de classes UML, version 1.5 ou plus.

6.2.1 Codage des classes

Les classes du diagramme de classes UML représentent les objets formels du contexte des classificateurs. Les attributs formels se composent des caractéristiques de la classe dans le méta-modèle, à savoir, `name`, `visibility`, `isAbstract`, `isActive`, `isDataType` et `isClass`. La Table 1 de l'Annexe B donne la signification des ces caractéristiques. La caractéristique `isActive`, par exemple, précise si une classe comporte un processus d'exécution propre. À noter que les instances d'une telle classe possèdent leur propre flot de contrôle et sont donc capables d'initier des activités. Chaque caractéristique de classe donne lieu à un attribut formel dans le contexte des classificateurs.

De plus, les attributs pluri-valués font l'objet d'un scaling conceptuel. Par exemple, l'attribut formel `name` dénote le nom de la classe et est pluri-valué du fait

de l'existence de plusieurs classes dans le diagramme. Par conséquent, cet attribut est échantillonné et engendre des attributs binaires de la forme `Name=className` (voir Table 6.3). Rappelons que les noms des classes sont codés dans le contexte des classificateurs par l'attribut formel pluri-valué `Name` afin d'éviter de faire la fusion entre classes ayant une description identique.

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

TAB. 6.3 – Contexte binaire des classes du diagramme de classes de la Figure 6.1 avec le scaling conceptuel de l'attribut formel pluri-valué `Name`.

À noter que si l'on considère le méta-modèle UML, les classes possèdent d'autres caractéristiques moins utiles pour le processus d'abstraction telles que `isLeaf`, `isRoot`, `isMain`, etc. De notre part, nous les avons omis par mesure de clarté de l'exemple. Les attributs, les opérations et les associations qu'une classe possède sont représentées par le biais des relations au sein de la FCR.

6.2.2 Codage des attributs

Les attributs en UML représentent les variables (de classe) stockant les informations d'état de l'objet. Dans les récentes versions de UML, un attribut est décrit par les propriétés suivantes : `name`, `visibility`, `initialValue`, `multiplicity`, `changeability`, `ownerScope` et `targetScope`. La Table 2 et la Table 3 de l'An-

nexe B donnent la signification de ces propriétés. La caractéristique ownerScope, par exemple, spécifie si chaque instance de la classe possède sa propre occurrence de l'attribut ou bien toutes les instances de la classe partage une seule occurrence de cet attribut (un attribut static en Java). Le contexte des attributs au sein de la FCR associe à chacune de ces propriétés un attribut formel. Ici aussi les attributs formels pluri-valués sont échantillonnés de manière conceptuelle.

Par exemple, le scaling de l'attribut formel visibility donne des attributs mono-valués de la forme visibility.public, visibility.private, visibility .protected et visibility.package. L'attribut ownerScope est échantillonné de façon nominale et donne les attributs ownerScope.instance et ownerScope.classifier. La Table 6.4 montre le codage des attributs du modèle UML de la Figure 6.1.

	Attributes								
	Name=number	Name=balance	Name=interestRate	InitialValue=null	Visibility=private	Multiplicity=[1,1]	Changeability=changeable	OwnerScope=instance	TargetScope=instance
CA.number	X			X	X	X	X	X	X
CA.balance		X		X	X	X	X	X	X
MA.number	X			X	X	X	X	X	X
MA.balance		X		X	X	X	X	X	X
MA.interestRate			X	X	X	X	X	X	X

TAB. 6.4 – Contexte binaire des attributs du diagramme de classes de la Figure 6.1.

6.2.3 Codage des opérations

Les opérations d'un objet caractérisent son comportement, c'est-à-dire l'ensemble des actions que l'objet peut réaliser pour réagir à des sollicitations extérieures

ou pour agir sur d'autres objets. En UML, une opération est documentée par les propriétés suivantes : `name`, `visibility`, `isAbstract`, `concurrency`, `isQuery` et `ownerScope`. La propriété `isQuery`, par exemple, précise si l'exécution de l'opération modifie l'état du système. La Table 4 de l'Annexe B montre la signification des propriétés de l'opération dans le méta-modèle UML alors que la Table 6.5, à gauche, montre le codage des opérations du modèle UML de la Figure 6.1.

Operations						
	Name=credit	Name=calculateInterest	Name=debit	Visibility.public	Concurrency.sequential	OwnerScope.instance
MA.credit()	X			X	X	X
MA.calculateInterest()		X		X	X	X
CA.credit()	X			X	X	X
CA.debit()			X	X	X	X

Associations	
	Name=owns
CBH-CA.owns	X
M-MA.owns	X

TAB. 6.5 – **Gauche** : le contexte binaire des opérations du diagramme de classes de la Figure 6.1. **Droite** : le contexte binaire des associations du diagramme de la même figure.

6.2.4 Codage des associations

Les associations dans le paradigme objet représentent les liens stables entre objets. En UML, une association possède un nom (`name`) et deux rôles (`association_End`). Le rôle est décrit dans le méta-modèle UML par les propriétés suivantes : `name`, `visibility`, `ordering`, `multiplicity`, `changeability`, `aggregation`, `targetScope` et `isNaviguable`. Cette dernière caractéristique, par exemple, une fois mise sur l'extrémité opposée, précise que l'association peut être parcourue en partant de la classe source et en utilisant le nom de rôle dans les expressions de navigation.

La Table 5 et la Table 6 de l'Annexe B donnent une description détaillée de ces propriétés. Par contre, la Table 6.5, à droite, et la Table 6.6 montrent le codage des associations et des rôles du modèle UML de la Figure 6.1. Comme les deux associations ont le même nom, les noms des objets formels respectifs sont obtenus par la concaténation de ce nom et d'un préfixe calculé à partir des noms des classes aux deux extrimités de chaque association.

À noter que les noms de rôles du modèle UML de la Figure 6.1 sont identiques et possèdent les mêmes valeurs pour certaines propriétés telles que la visibilité, les multiplicités, etc. Ceci ne constitue aucunement une obligation pour la méthode de restructuration de l'ARC.

Rôles d'associations									
	Name=owner	Name=account	Visibility=private	Multiplicity=[1,1]	Ordering=unordered	Aggregation=none	Changeability=changeable	TargetScope=instance	isNaviguable
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

TAB. 6.6 – Contexte binaire des extrémités d'associations du diagramme de classes de la Figure 6.1.

6.2.5 Relations inter-contextes

Une FCR est composée de contextes et de relations inter-contextes comme il a été expliqué dans le Chapitre 4, section 4.3. Dans le cas d'une FCR représentant un modèle de classes UML, les relations inter-contextes sont inspirées des associa-

tions entre les éléments du méta-modèle qui correspondent aux différents contextes. Comme le montre la Table 6.8, parmi les associations entre les éléments du méta-modèle, on trouve : `owned_attribute`, `owned_operation`, `type`, `return_type`, `belongs_to_association`, `first_end`, `second_end`, `owned_association_end` et `specified_class`. La Table 6.7 récapitule ces relations ainsi que le contexte source et celui de destination.

Relation	Contexte source	Contexte de destination
`owned_attribute`	`Classificateurs`	`Attributs`
`owned_operation`	`Classificateurs`	`operations`
`type`	`Attributs`	`Classificateurs`
`return_type`	`Operations`	`Classificateurs`
`first_param`	`Operations`	`Classificateurs`
`second_param`	`Operations`	`Classificateurs`
`belongs_to_association`	`Rôles`	`Associations`
`first_end`	`Associations`	`Rôles`
`second_end`	`Associations`	`Rôles`
`owned_association_end`	`classificateurs`	`Rôles`
`specified_class`	`Rôles`	`Classificateurs`

TAB. 6.7 – Tableau recapitulatif des relations inter-contextes d'une FCR codant un modèle de classe UML.

La relation `owned_attribute` associe à chaque classe du modèle la liste des attributs qui lui sont rattachés. La Table 6.8, à gauche, donne l'état de cette relation pour le modèle UML de la Figure 6.1.

De manière similaire, la relation `type` associe à chaque attribut du modèle un type de données qui représente son domaine de valeurs ou une classe du modèle. La Table 6.8, à droite, donne l'état de cette relation pour le modèle UML de la Figure 6.1.

Les relations `owned_operation` et `return_type` relient les deux contextes, celui des classes et celui des opérations. La première relation fait correspondre à chaque classe ses propres opérations déclarées alors que la deuxième relation précise pour chaque opération, le type de la valeur retournée. La Table 6.9 montre l'état de ces deux relations dans le cas du modèle UML de la Figure 6.1.

owned_attribute (OA)					
	CA.number	CA.balance	MA.number	MA.balance	MA.interestRate
CA	X	X			
MA			X	X	X
CBH					
M					

type (TY)						
	CA	MA	M	CBH	String	Double
CA.number					X	
CA.balance						X
MA.number					X	
MA.balance						X
MA.interestRate						X

TAB. 6.8 – **Gauche** : la relation qui associe chaque classe à ses attributs. **Droite** : la relation qui fait correspondre chaque attribut à sa classe ou son type de données.

owned_operation (OO)				
	MA.credit()	MA.calculateInterest()	CA.credit()	CA.debit()
CA			X	X
MA	X	X		
CBH				
M				

return_type (RT)						
	Double	String	CA	MA	M	CBH
MA.credit()	X					
MA.calculateInterest()	X					
CA.credit()	X					
CA.debit()	X					

TAB. 6.9 – **Gauche** : la relations Classificateurs × Operations. **Droite** : La relations Operations × Classificateurs.

La relation `belong_to_association` code les liens qui existent entre les extrémités des associations (rôles) et les associations du modèle UML. La Table 6.10 donne l'état de cette relation dans le cas du modèle UML de la Figure 6.1.

belong_to_association (BTA)		
	CBH-CA.owns	M-MA.owns
CBH-CA.owns.owner	X	
CBH-CA.owns.account	X	
M-MA.owns.owner		X
M-MA.owns.account		X

TAB. 6.10 – Relation entre les associations et les rôles d'association.

Les relations `first_end` et `second_end` codent les liens qui existent entre les associations et leurs extrémités. Elles précisent pour chaque association la première et la deuxième extrémité. La Table 6.11 décrit ces deux relations dans le cas du modèle UML de la Figure 6.1.

first_end (FE)				
	CBH-CA.owns.owner	CBH-CA.owns.account	M-MA.owns.owner	M-MA.owns.account
CBH-CA.owns	X			
M-MA.owns			X	

second_end (SE)				
	CBH-CA.owns.owner	CBH-CA.owns.account	M-MA.owns.owner	M-MA.owns.account
CBH-CA.owns		X		
M-MA.owns				X

TAB. 6.11 – Les relations entre les associations et les rôles d'association (suite).

Maintenant, reste à coder les liens entre les classificateurs et leurs rôles respectifs. C'est ce que les relations `owned_association_End` et `specified_class` proposent de faire. En effet, la relation `owned_association_End` précise, pour chaque classe, les rôles qui lui sont rattachés alors que la relation `specified_class` opère de manière inverse, c'est-à-dire, précise pour chaque rôle la classe désignée. La Table 6.12 décrit ces deux relations dans le cas du modèle UML de la Figure 6.1.

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		

TAB. 6.12 – Les relations entre les classes et les rôles d'associations.

6.3 Contrôle de la sémantique du modèle

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 dont la sémantique est identique et inversement.

6.3.1 Résolution des conflits de noms

Les éléments d'un modèle UML risquent de se chevaucher dans leurs noms ou leurs descriptions (propriétés et liens vers d'autres éléments). En effet, bien qu'ils fournissent beaucoup de rigueur et de structuration, les modèles UML, comme d'autres langages de description de hiérarchies, sont essentiellement basés sur le langage naturels et par conséquent potentiellement ambigus. Évidemment, une correspondance parfaite entre noms et descriptions est une bonne indication pour

des éléments représentant un même concept. Le chevauchement est un aspect de conflits difficile car dans certains cas il peut être la manifestation de phénomènes linguistiques complexes tels que la *synonymie* (à une même entité sont rattachés différents noms) et la polysémie (un même nom dénote différentes entités).

À noter que ces problèmes ont été considérés dans le cadre de l'intégration de schémas de base de données et la solution standard habituellement suggérée est 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 [6, 67]. Il est clair que l'approche basée sur la connaissance linguistique toute seule n'est ni universellement applicable, ni capable de permettre à un outil de résoudre tous les conflits de manière strictement automatique. Par conséquent, l'intervention d'un expert humain est toujours indispensable pour valider la solution proposée. En génie logiciel, la situation s'est nettement améliorée avec l'avènement des langages/méthodes de modélisation mettant l'emphase sur la mise en place d'un espace de noms unique pour tous les éléments du modèle. Ceci permet de réduire la disparité de noms. Néanmoins, un outil automatique réaliste devrait pouvoir détecter les situations suspectes dans un modèle où des conflits potentiels de noms résident.

Les propriétés structurales du treillis et de la position de ces éléments dans celui-ci peuvent aussi servir au même but, c'est à dire, résoudre de possibles conflits de nommage. Étant données deux termes suspects, un outils de résolution de conflits sophistiqués peut davantage montrer les conséquences de l'impact d'une action particulière sur ces termes, notamment la fusion des deux termes, la création d'une généralisation dédiée, le découpage en termes composants (tokenization).

En somme, une approche de résolution de conflits entièrement manuelle pourrait être basée sur le mécanisme d'*exploration des attributs* [30] (attribute exploration) qui est fondamentalement une méthode d'acquisition de connaissances assistée par un outil automatique dialoguant avec le concepteur expert. L'outil oriente le concepteur du modèle à la découverte d'un ensemble minimal d'implications qui représentent la totalité du domaine tout en réduisant au minimum l'interaction

outil-expert. Il faut noter que les outils AFC mentionnés ci-dessus tels que l'extraction des implications et l'exploration des attributs doivent encore être étendus aux descriptions relationnelles qui sont traitées ici dans le cadre de l'ARC.

6.4 Construction des treillis relationnels

À partir d'une FCR, la méthode MULTI-FCA permet de construire une famille de treillis relationnels (FTR) où les liens inter-individus sont abstraits par des relations inter-concepts. L'exécution de cette méthode avec la FCR codant le modèle de classes de la Figure 6.1 s'est déroulée de la manière suivante :

6.4.0.0.1 L'étape d'initialisation : à l'étape d'initialisation, les treillis des différents contextes sont construits. Les treillis de la Figure 6.9, Figure 6.10, Figure 6.11 et Figure 6.12 correspondent au contexte des classificateurs, attributs, opérations et rôles, respectivement.

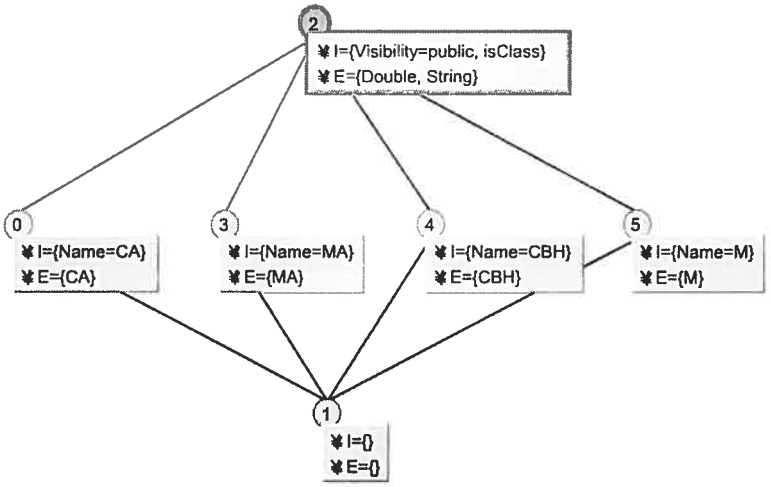


FIG. 6.9 – Le treillis initial du contexte des classificateurs (correspond au contexte de la Table 6.3).

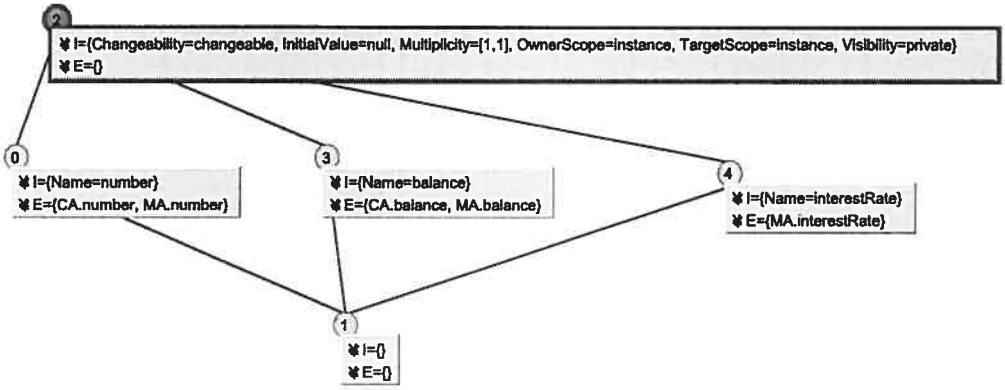


FIG. 6.10 – Le treillis initial du contexte des attributs (correspond au contexte de la Table 6.4).

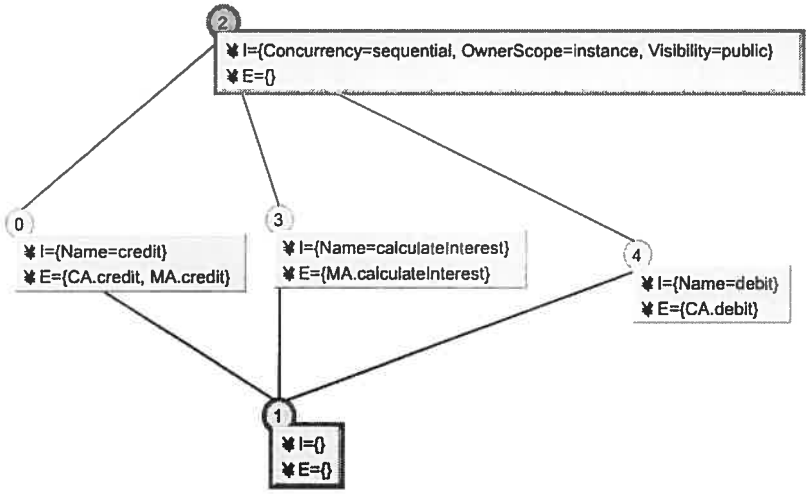


FIG. 6.11 – Le treillis initial du contexte des opérations (correspond au contexte illustré à gauche de la Table 6.5, à gauche).

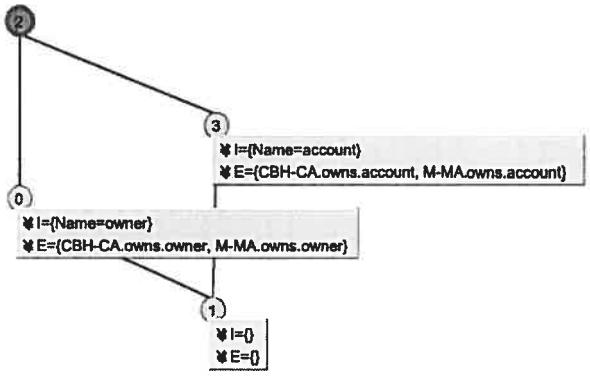


FIG. 6.12 – Le treillis initial du contexte des rôles (correspond au contexte de la Table 6.6).

En considérant les différents treillis initiaux, plusieurs abstractions ont été déjà découvertes pour chacun des types UML représentés. Par exemple, le concept $c_{\#3}$ de la Figure 6.10 représente les similarités entre l'attribut `balance` de la classe `CheckAccount` et celui de la classe `MortgageAccount` tel que le nom, la visibilité, la multiplicité tandis que le concept $c_{\#0}$ de la Figure6.12 représente les similarités entre le rôle `owner` de la classe `Mortgager` et le rôle `owner` la classe `CheckBookHolder` notamment de rôle, etc.

6.4.0.0.2 La première itération : Ensuite, au cours de la première étape du processus itératif MULTI-FCA, chaque contexte est enrichi en se basant sur les concepts des treillis initiaux des contextes auquels il est relié par une relation. En effet, comme expliqué dans Chapitre 4, Section 4.5.1, pour une relation donnée, certains concepts[2] du treillis associé au contexte du co-domaine de cette relation deviennent des attributs formels dans le contexte du domaine de la même relation. Par exemple, le contexte des classificateurs est relié au contexte des attributs par la relation `owned_attribute` (`OA`). Ainsi, les concepts $c_{\#0}$, $c_{\#2}$, $c_{\#3}$ et $c_{\#4}$ du

[2]Étant donnée une relation $r : \mathcal{K}_i = (O_i, A_i, I_i) \rightarrow \mathcal{K}_j$, $\forall o \in O_i$, $\forall c \in \mathcal{L}_j$, $(o, c) \in I_i \Leftrightarrow r(o) \cap Extent(c) \neq \emptyset$.

treillis des attributs de la Figure 6.10 sont utilisés pour former de nouveaux attributs formels dans le contexte des classificateurs (voir Table 6.13, colonnes 7-10). Le concept $c_{\#1}$ du même treillis est ignoré car son extension est égale à l'ensemble vide et par conséquent ni le codage étroit ni le large ne permettent de le considérer dans le processus du scaling de la relation `owned_attribute`(voir Chapitre 4, Section 4.5.1.2). Le contexte des classificateurs est aussi relié au contexte des rôles par le biais de la relation `owned_association_end` (`OAE`) et par conséquent les concepts du treillis initial des rôles est utilisé pour coder cette relation dans le contexte des classificateurs.

La Table 6.13 et la Figure 6.13 donnent l'état du contexte des classificateurs après le scaling de toutes ses relations et le treillis correspondant, respectivement.

Suite à cet enrichissement, le treillis des classificateurs nous conduit à la découverte de plusieurs nouvelles classes qui factorisent les propriétés des classes initiales du modèle. Par exemple, le concept $c_{\#6}$ représente une nouvelle classe, qu'on peut appeler `Account`, qui généralise les deux types de comptes `CheckAccount` et `MortgageAccount`.

Codage initial							Extension de la 1^{ere} itération									
	Name=CA	Name=MA	Name=CBH	Name=M	Visibility=public	isClass	OA:c2	OA:c0	OA:c3	OA:c4	OO:c2	OO:c0	OO:c4	OO:c3	OAE:c2	OAE:c3	OAE:c0
CheckAccount (CA)	X				X	X	X	X	X		X	X	X		X	X	
MA		X			X	X	X	X	X	X	X	X		X	X	X	
CBH			X		X	X									X		X
M				X	X	X									X		X
String					X	X											
Double					X	X											

TAB. 6.13 – Le contexte des classificateurs du modèle UML de la Figure 6.1 après échantiollonnage de toutes les relations au bout de la première itération de l'algorithme MULTI-FCA.

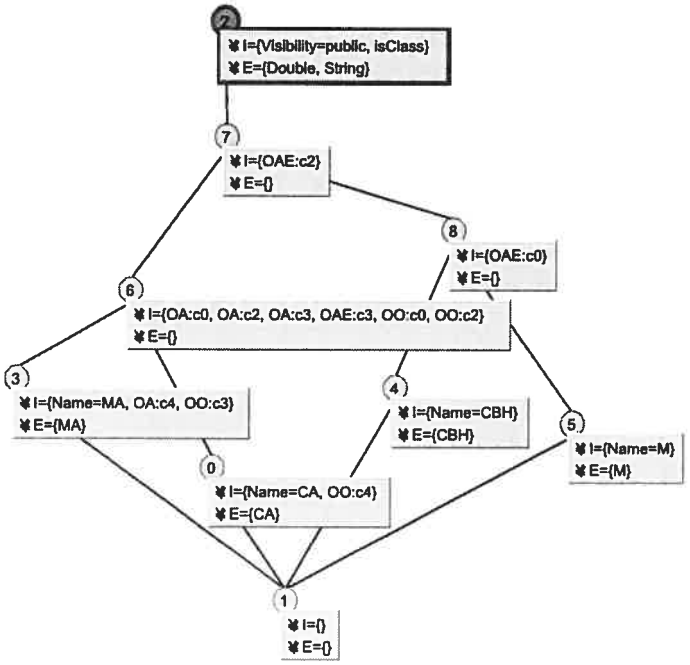


FIG. 6.13 – Le treillis des classes (correspond au contexte de la Table 6.13) au bout de la première itération.

L'interprétation du concept $c_{\#6}$ donne la description en cours[3] de la classe `Account` dont les propriétés sont représentées par les attributs formels relationnels suivants : `OA:c0`, `OA:c3`, `OO:c0`, et `OAE:c3`. Les attributs relationnels `OA:c0` et `OA:c3` correspondent aux variables de classe `number` et `balance` selon le treillis des attributs donné par la Figure 6.10 (voir les concepts $c_{\#0}$ et $c_{\#3}$). L'attribut relationnel `OO:c0` correspond à l'opération `credit` selon le concept $c_{\#0}$ du treillis des opérations de la Figure 6.11. Finalement, l'attribut relationnel `OAE:c3` correspond à une nouvelle généralisation de rôle décrite par le concept $c_{\#3}$ du treillis des rôles de la Figure 6.12. Cette généralisation que nous allons appeler `account` est réalisée à partir des deux rôles de base `CBH-CA.owns.account` et `M-MA.owns.account`.

La traduction du treillis des classificateurs de la première étape du processus

[3]Succeptible d'augmenter au cours des prochaines itérations.

itératif en classes et rôles est illustrée par la Figure 6.14. À noter que le diagramme de classes que représente cette figure est incomplet car à ce stade du processus itératif les informations sur les associations ne sont pas encore disponibles. Nous avons aussi précisé le numéro du concept dans le treillis correspondant de chaque élément du diagramme de classes.

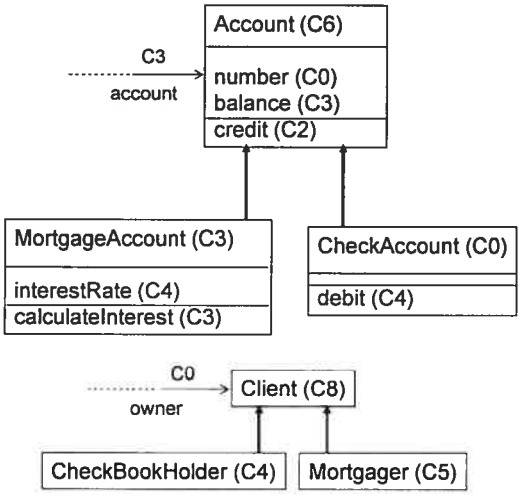


FIG. 6.14 – Traduction en classes et rôles du treillis des classificateurs de la première itération (Figure 6.13). Deux associations ne sont pas encore connues car une seule extrémité est déterminée pour chacune.

De façon similaire, le contexte des rôles d'associations a été enrichi par de nouveaux attributs relationnels en provenance des treillis des classificateurs et celui des associations grâce aux relations `specified_class` (SC) et `belongs_to_association` (BTA). La Table 6.14 et la Figure 6.15 montrent le contexte enrichi et son treillis, respectivement.

On remarque sur le treillis des rôles de la Figure 6.12 que les deux rôles `CBH-CA .owns.account` et `M-MA.owns.account` appartiennent au même concept-objet car ils ont une description similaire. L'extension relationnelle du contexte des rôles obtenue au bout de la première étape du calcul itératif les a séparé au niveau du treillis (voir Figure 6.15) car il n'ont pas les mêmes liens avec les classes (re-

Codage initial										Ext. 1^{ere} itération					
	Name=owner	Name=account	Visibility=private	Multiplicity=[1,1]	Ordering=unordered	Aggregation=none	Changeability=changeable	TargetScope=instance	isNavigable	SC:c2	SC:c4	SC:c0	SC:c5	SC:c3	BTA:c0
CBH-CA.owns.owner	X		X	X	X	X	X	X	X	X	X				X
CBH-CA.owns.account		X	X	X	X	X	X	X	X	X		X			X
M-MA.owns.owner	X		X	X	X	X	X	X	X	X			X		X
M-MA.owns.account		X	X	X	X	X	X	X	X	X				X	X

TAB. 6.14 – Contexte binaire des rôles du diagramme de classes de la Figure 6.1 au bout de la première étape du processus itératif MULTI-FCA.

lation specified_class) et n'appartiennent pas à la même association (relation belongs_to_association).

L'interprétation du treillis des rôles de la première étape du processus itératif en un diagramme de classes est donnée par la Figure 6.16. On remarque l'émergence de deux nouvelles abstractions de rôles dont les classes spécifiées (relation specified_class) ne sont pas encore connues. Ces abstractions correspondent aux concepts $c_{\#0}$ (owner) et $c_{\#3}$ (account).

Finalement, les autres contextes ont subi aussi un enrichissement mais leurs treillis respectifs sont restés isomorphes aux treillis initiaux.

6.4.0.0.3 La deuxième itération : Lors de cette itération, les extensions relationnelles des différents contextes sont recalculées[4] en utilisant les treillis obtenus au bout de la première itération comme échelles pour les différentes relations. La Table 6.15 donne l'état du contexte des classificateurs après le scaling de toutes

[4]Comme les treillis sont obtenus par des opérations de mise-à-jour et que ces opérations sont monotones, une mise-à-jour des contextes est suffisante.

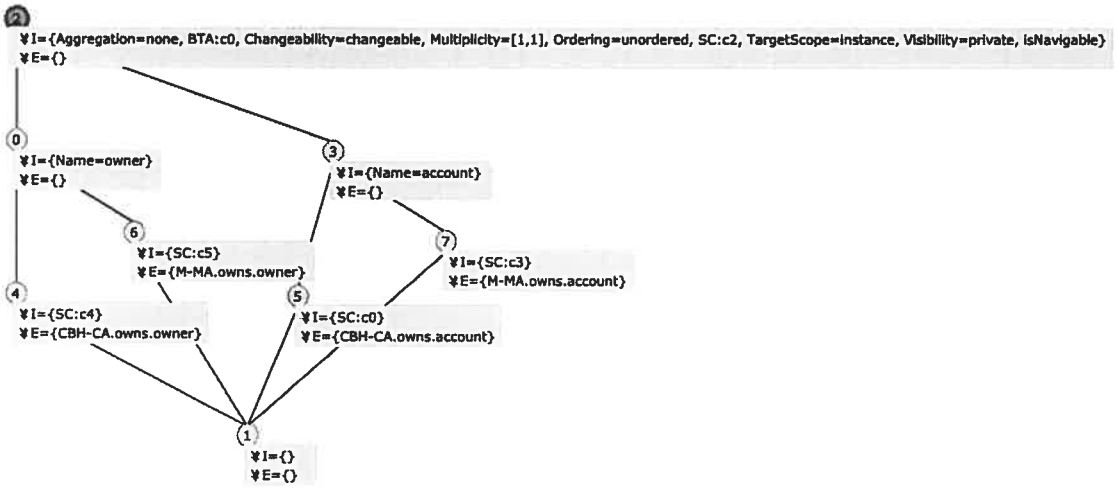


FIG. 6.15 – Le treillis des rôles (correspond au contexte de la Table 6.14) obtenu au bout de la première itération.

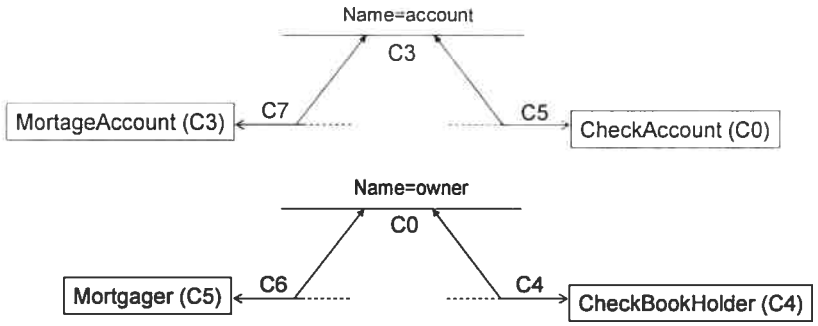


FIG. 6.16 – Traduction en classes et rôles du treillis des rôles de la première itération (Figure 6.15). On ne sait pas, à ce niveau du processus itératif, les classes auxquelles appartiennent les nouvelles abstractions de rôles.

ses relations. On remarque, par exemple, l'intégration des nouveaux concepts-rôles $c_{\#4}$, $c_{\#5}$, $c_{\#6}$ et $c_{\#7}$ découverts au bout de l'itération précédente comme attributs relationnels pour la relation `owned_association_end` (`OAE`).

Codage initial		Ext. 1^{ere} itération											Ext. 2^{eme} itération			
	...	OA:c2	OA:c0	OA:c3	OA:c4	OO:c2	OO:c0	OO:c4	OO:c3	OAE:c2	OAE:c3	OAE:c0	OAE:c5	OAE:c7	OAE:c4	OAE:c6
CA	...	X	X	X		X	X	X		X	X		X			
MA	...	X	X	X	X	X	X		X	X	X			X		
CBH	...									X		X			X	
M	...									X		X				X
String	...															
Double	...															

TAB. 6.15 – Le contexte des classificateurs du modèle UML de la Figure 6.1 au bout de la seconde et dernière itération.

La Figure 6.17 illustre le nouveau treillis des classificateurs. On remarque que le concept $c_{\#0}$ (`CheckAccount`) et $c_{\#3}$ (`MortageAccount`) désignent maintenant les deux concepts-rôles $c_{\#5}$ et $c_{\#7}$ (rôle `account`) alors que leur super-concept $c_{\#6}$ (classe `Account`) désigne le super-concept de $c_{\#5}$ et $c_{\#7}$, le concept $c_{\#3}$ (rôle `account`) du treillis des rôles.

La Figure 6.18 donne l'interprétation en un diagramme de classes du treillis des classificateurs de la seconde itération. On remarque que les abstractions des rôles de la première itération on été intégrées au modèle de classes car désormais la classe `Account` possède le rôle `account` fraîchement découvert.

De retour au contexte des rôles, la nouvelle extension relationnelle calculée à partir des treillis de la première itération est donnée par la Table 6.16. Le treillis associé est illustré par la Figure 6.19. On remarque la présence de l'attribut relationnel `SC :c6` dans l'intension du concept-rôle $c_{\#3}$ (rôle `account`). Cet attribut n'était pas présent au niveau du treillis des rôles de la première itération et indique le concept-classificateur $c_{\#6}$ (classe `Account`). Cela signifie que la nouvelle classe `Account` possède un rôle dont le nom est `account`.

La Figure 6.20 illustre l'interprétation en un diagramme de classes du treillis des

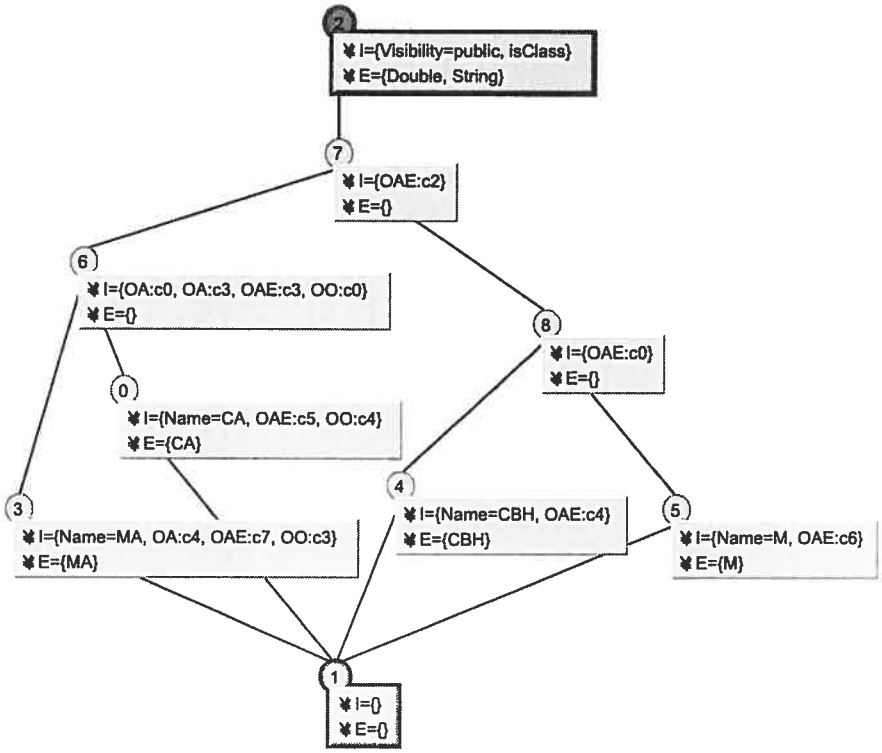


FIG. 6.17 – Le treillis des classificateurs (correspond au contexte de la Table 6.15) au bout de la deuxième et dernière itération.

Codage initial		Ext. 1^{ere} itération						Ext. 2^{eme} itération		
	...	SC:c2	SC:c4	SC:c0	SC:c5	SC:c3	BTA:c0	SC:c7	SC:c8	SC:c6
CBH-CA.owns.owner	...	X	X				X	X	X	
CBH-CA.owns.account	...	X		X			X	X		X
M-MA.owns.owner	...	X			X		X	X	X	
M-MA.owns.account	...	X				X	X	X		X

TAB. 6.16 – Intégration des classes `Account` (SC:c6) et `Client` (SC:c8) fraîchement découvertes à l'extension relationnelle du contexte des rôles au bout de la seconde étape du processus itératif MULTI-FCA.

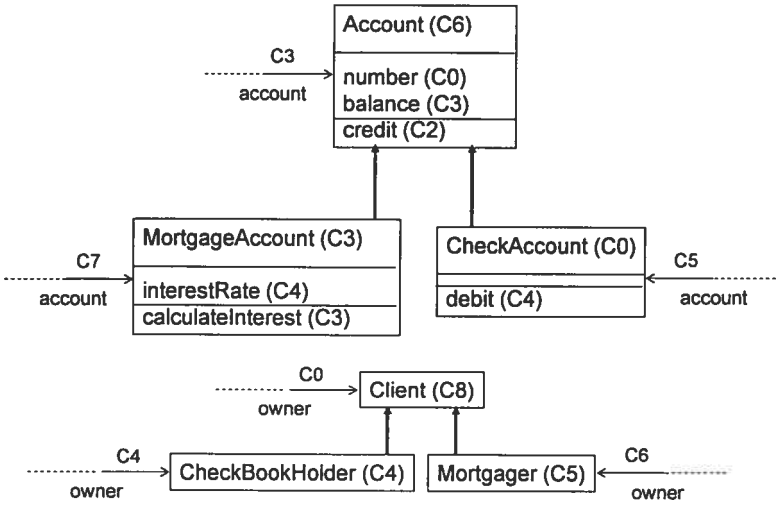


FIG. 6.18 – Traduction en classe et rôles du treillis des classificateurs de la seconde itération (Figure 6.17).

rôles. On constate le raffinement du modèle de classes par, d'une part la précision des détails concernant les rôles du modèle initial tel que le rôle induit par le concept-rôle $c_{\#7}$ qui maintenant précise la classe à laquelle il appartient ; et d'autre part, la découverte de deux nouvelles abstractions de rôles, à savoir `account` (concept-rôle $c_{\#3}$) et `owner` (concept-rôle $c_{\#0}$).

La méthode MULTI-FCA s'arrête à ce niveau car les treillis obtenus au niveau de la prochaine itération sont tous isomorphes à ceux de la seconde itération. Sur le plan de la modélisation, aucune nouvelle généralisation n'est découverte. Dans le paragraphe suivant, nous allons donner une approche permettant de générer le modèle UML à partir de la famille de treillis finaux obtenue par le processus MULTI-FCA.

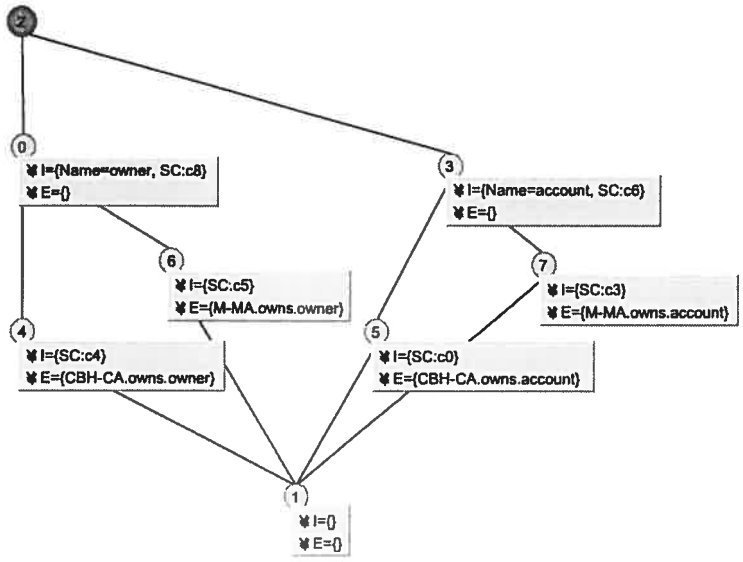


FIG. 6.19 – Le treillis des rôles (correspond au contexte de la Table 6.16) au bout de la deuxième et dernière itération. Le concept $c_{\#2}$=(∅,{aggregation=none, BTA:c0,changeabuility=changeable,visibility=[1,1],Ordering=unordered, SC:c2,SC:c7,TargetScope=instance, Visibility=private,isNavigable}).

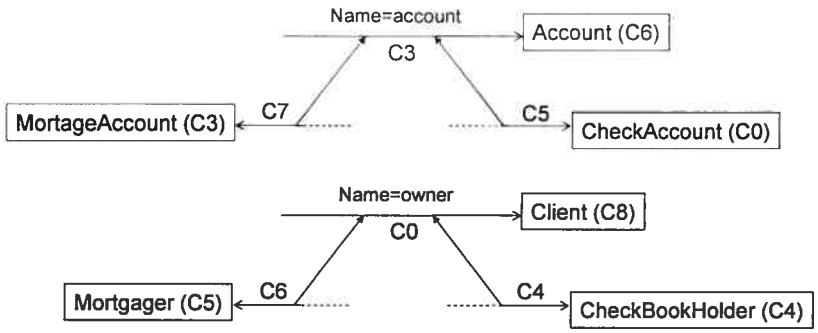


FIG. 6.20 – Traduction en classes et rôles du treillis des rôles de la seconde itération (Figure 6.19).

6.5 Rétro-codage du modèle UML

Le treillis final des classificateurs (voir Figure 6.17) constitue le point de départ de l'opération de génération du diagramme de classes final dont l'une des qualités principales, rappelons le, est l'introduction d'abstractions significatives sur l'ensemble des éléments de modélisation ainsi que la factorisation maximale des propriétés de ces éléments. Les concepts du treillis des classificateurs, appelés aussi concepts-classificateurs, représentent toutes les abstractions possibles des classificateurs obtenues à partir du partage des attributs, méthodes et rôles. Dans le language de l'ARC, ces concepts ont été obtenus suite à l'extension relationnelle du contexte initial des classificateurs par les relations dont ce contexte est le domaine dans la FCR du modèle UML initial, à savoir, `Owned_Attribute` (`OA`), `Owned_Operation` (`OO`) et `Owned_Association_End` (`OAE`). Par exemple, l'intension simplifiée du concept-classificateur $c_{\#3}$ de la Figure 6.17 est composée de : `name=MA`, `OA:c4`, `OAE:c7` et `OO:c3`. Ceci signifie que ce concept-classificateur est lié au concept-attribut-formel $c_{\#4}$ du treillis des attributs (voir Figure 6.10) par la relation `Owned_Attribute`, au concept-rôle $c_{\#7}$ du treillis des rôles (voir Figure 6.19) par la relation `Owned_Association_End` et finalement au concept-opération $c_{\#3}$ du treillis des opérations (voir Figure 6.11) par la relation `Owned_Operation`. L'interprétation en UML du concept-classificateur $c_{\#3}$ est une classe appelée `MortgageAccount` possédant l'attribut `interestRate`, le rôle `account` et l'opération `calculateInterest` (voir Figure 6.22).

Dans ce qui suit, nous allons présenter une approche de détection des abstractions significatives des éléments du modèle UML initial en exploitant les treillis finaux des différents contextes codant le modèle UML initial.

6.5.1 Pertinence d'un concept

Tout concept peut être traduit en un élément UML candidat pour le modèle final. Toutefois, les treillis finaux sont en général de grande taille et comportent beaucoup de concepts qui n'ont pas d'importance sur le plan de la modélisation

du domaine. Il faut donc procéder à une selection des concepts candidats ayant un intérêt potentiel pour les objectifs de modélisation élaborés par le concepteur. Pour ce faire, un calcul de pertinence est mis en place.

Par exemple, le concept-classificateur $c_{\#2}$=(`visibility=public`, `isClass`) n'est pas traduit en une classe car il n'est pas pertinent selon les critères de pertinence que nous allons définir par la suite, tandis que le concept-classificateur $c_{\#6}$=(`OA:c0`, `OA:c2`, `OA:c3`, `OAE:c3`, `OO:c0`, `OO:c2`) est traduit en une classe, appelée `Account` dans le modèle de classes UML généré (voir la Figure 6.22) dont les attributs sont : `number` (`OA:c0`) et `balance` (`OA:c3`), l'unique opération est `credit` (`OO:c0`) et un rôle `account` (`OAE:c3`). Les attributs relationnels `OA:c2` et `OO:c2` ont été abondonnés car tous deux correspondent, comme nous allons le voir dans la Section 6.5.3, à des abstractions non significatives dans leurs treillis respectifs.

Tout d'abord, il faut distinguer entre deux types d'information disponibles dans la FCR codant le modèle UML, à savoir, l'information '*métier*' et l'information '*technique*'. L'information métier provient du domaine du modèle et est représentée par les éléments du modèle. Cette information est codée par les individus des différents contextes tels que les attributs, les rôles, les opérations, etc. ainsi que les liens inter-individus tels que `owned_attribute`, `owned_association_end`, etc. L'information technique provient des propriétés des éléments du modèle et est définie dans le méta-modèle UML telles que la visibilité, les multiplicité, la navigabilité, la portée, etc. Cette information est représentée dans la FCR par le biais des attributs formels décrivant les différents types d'individus.

Les concepts, tous types confondus, n'ayant dans leurs intensions que des informations techniques sont écartés lors du processus de selection des concepts pertinents car il est peu utile, sur le plan de la modélisation, d'avoir des abstractions de classes ou de rôles représentant, disons, toutes les classes dont la visibilité est 'public' ou des rôles navigables, respectivement. Par exemple, le concept-classificateur $c_{\#2}$ du treillis des classificateur illustré par la Figure 6.17 dont l'intension simplifiée est {`visibility=public`, `isClass`} représente toutes les classes du modèle dont la visibilité est 'public'.

Ainsi, un concept est jugé pertinent, c'est à dire donnant lieu à une abstraction potentiellement intéressante une fois interprété en UML, si son intension renferme au moins un attribut formel de nature métier. Par exemple, le concept-rôle $c_{\#3}$ du treillis des classificateurs (voir Figure 6.17) dont l'intension simplifié est {`name=MA`,`OA:c4`,`OAE:c7`,`OO:c3`} renferme l'information métier `name=MA` qui fait référence à la classe `MortageAccount` du modèle UML initial.

La pertinence d'un concept ayant dans son intension des attributs relationnels est obtenue de façon transitive en examinant la pertinence des concepts désignés. Toutefois, ces attributs relationnels doivent être examinés dans un ordre tel que :

- Concept-classificateur : on évalue la pertinence suivant l'ordre suivant : les attributs, les opérations et finalement les rôles.
- Concept-rôle : on évalue la pertinence suivant les classificateurs ensuite les associations.
- Concept-association : ce type de concept peut disposer de deux types d'attributs relationnels obtenus à partir des deux relations `first_end` et `second_end` et peu importe l'ordre dans lequel ces attributs relationnels sont évalués.
- Concept-attribut-formel ou concept-opération : ces deux types de concepts ne peuvent avoir qu'un seul type d'attribut relationnel obtenu à partir de la relation `type` ou `returnType`, respectivement.

Les attributs relationnels désignant des concepts non pertinents ne sont pas considérés lors de la traduction en UML du concept propriétaire.

La définition recursive de la pertinence d'un concept pose un problème de circularité dans le calcul de cette pertinence. C'est le cas par exemple de deux concepts inter-dépendants qui ne représentent pas d'information du domaine. Les concepts inter-dépendants émergent à partir de contextes inter-dépendents tels que le contexte des classificateurs et celui des rôles d'associations qui sont inter-reliés par les relations `owned_assoc_end` et `specified_class`. Le concept-classificateur $c_{\#7}$ (Figure 6.17), le concept-rôle $c_{\#2}$ (Figure 6.19) et le concept-association $c_{\#0}$ ont les intensions simplifiées suivantes :

- $Int(c_{\#7})$={`OAE:c2`},

- $Int(c_{\#2})$={aggregation=none, BTA:c0, changeability=changeable, multiplicity=[1,1], ordering=unordered, SC:c2, SC:c7, visibility=private, targetscope=instance, isNavigable},
- $Int(c_{\#0})$={FE:c2, SE:c2},

Ainsi, le concept-classificateur $c_{\#7}$ est relié au concept-rôle $c_{\#2}$ par la relation owned_assoc_end (OAE:c2), le concept-rôle $c_{\#2}$ est relié au concept-classificateur $c_{\#7}$ par la relation specified_class (SC:c7) et au concept-association $c_{\#0}$ par la relation Belong_to_association (BTA:c0) et finalement le concept-association $c_{\#0}$ est relié au concept-rôle $c_{\#2}$ par les les deux relations first_end (FE:c2) et second_end (SE:c2). De plus, les trois concepts sont techniques car leurs intensions respectives ne possèdent aucun autre attribut métier. La Figure 6.21 illustre les dépendances entre ces deux concepts.

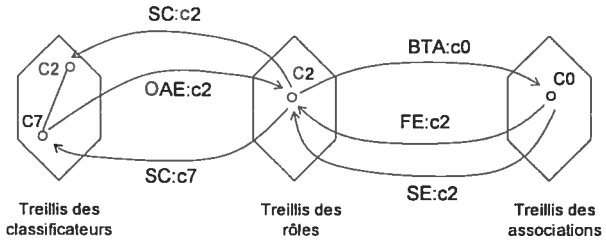


FIG. 6.21 – Relations circulaires entre les concepts de la FTR.

Il faut donc envisager des mécanismes permettant de casser la circularité et de se pronencer sur la pertinence des concepts impliqués. Pour notre part, tous les concepts dont l'évaluation de la pertinence est confrontée au problème de la circularité des relations sont soumis au jugement de l'utilisateur.

Rappelons enfin que l'information technique disponible avec un concept pertinent est très utile car elle permet d'avoir tous les détails techniques nécessaires à la traduction directe du concept en question en un élément UML.

L'algorithme 7 permet de déterminer si un concept est pertinent en s'appuyant sur la routine DOMAIN qui vérifie si un attribut formel donné désigne une information qui fait partie du domaine du modèle (ligne 10). Afin de surveiller la

circularité au niveau des concepts inter-dépendants, l'algorithme utilise la structure de données `Path` pour détecter la présense d'un même concept plus qu'une fois sur un même chemin de calcul de pertinence.

Étant donné un concept c, l'agorithme 7 commence par le considérer comme étant non pertinent (ligne 4). Ensuite, si ce concept a été déjà visité, l'algorithme s'arrête en retournant une valeur indiquant l'impossibilité d'évaluer sa pertinence (lignes 5 et 6) ; sinon, le concept est rajouté au chemin de pertinence (ligne 8). À ce stade, l'algorithme parcourt de façon séquentielle tous les attributs formels de l'intension du concept c (ligne 9) et pour chaque attribut vérifie d'abord s'il fait partie des attributs formels locaux, notés $Int_l(c)$ (ligne 10) ensuite s'il appartient au domaine du modèle (ligne 11). Si c'est le cas, l'algorithme s'arrête (ligne 12) en considérant le concept comme étant pertinent par rapport au domaine. Les attributs qui ne sont pas du domaine mais qui sont de type relationnel, notés $Int_r(c)$, subissent une évaluation de pertinence qui affecte la pertinence du concept courant c (ligne 14). Si cette dernière évaluation s'avère positive, le concept c est déclaré pertinent par transitivité car il indique un concept pertinent et il n'est plus utile d'examiner les attributs formels non visités de son intension (ligne 16).

6.5.1.0.4 **Routine Domain :** en examinant la manière dont les éléments métiers sont codés dans la FCR on remarque que ces éléments représentent les objets formels des différents contextes et que des attributs formels ont été créés sous la forme '`name=`*nom-élément*' dans les contextes respectifs. Le treillis de chaque type d'élément (classificateurs, attributs, rôles, etc.) possède un concept-objet par élément de même type qui code de manière complète toute l'information disponible dans le modèle initial sur cet élément. Par exemple, l'attribut métier `balance` de la classe `CheckAccount` (voir Figure 6.1) est codé dans le contexte des attributs illustré par la Table 6.4 par l'objet formel `CA.balance`. De plus, il y a un attribut formel `name=balance` appartenant à la description de l'objet formel `CA.balance`. Le concept $c_{\#3}$ du treillis des attributs illustré par la Figure 6.10 représente l'attribut métier `balance`. Par conséquent, pour vérifier si un attribut formel donné

```
1: Algorithm RELEVANT
2: (In : RLF a family of lattices, c a concept, Path chain of concepts) : [1..3]
3: Local   : relevant an integer variable
4: relevant ← 2        {by default classifier-concept is set to be irrelevant}
5: if c ∈ Path then
6:    return 3      {relevancy can't be determinated because of cycle in concept relations}
7: else
8:    Path ← Path ∪ {c}
9: for all a ∈ Int(c) do
10:   if a ∈ Int_l(c) then
11:      if DOMAIN(a) then
12:         return 1      {relevant classifier-concept}
13:   else
14:      relevant ← RELEVANT(RLF,cod(a),Path)      {a ∈ Int_r(c)}
15:      if relevant=1 then
16:         return 1      {relevant classifier-concept}
17: return relevant
```

Algorithm 7: Détermination de la pertinene d'un concept.

est un attribut métier il suffit à la routine DOMAIN de s'assurer que le nom de cet attribut est de la forme '`name`=*nom-élément*'.

Exemple : supposons que l'on veut calculer la pertinence du concept-classificateur $c_{\#5}$ (voir Figure 6.17) dont l'intension est égale à {`name=M, OAE:c6`}. Ce concept est jugé pertinent par l'algorithme 7 car l'intension renferme un attribut métier (préfixe `name`, lignes 10 et 11). Considérons maintenant le concept-classificateur $c_{\#6}$ dont l'intension est composée exlusivement d'attributs relationnels : {`OA:c0, OA:c2, OA:c3, OAE:c3, OO:c0,OO:c2`}. Supposons qu'on veut calculer la pertinence de ce concept à partir de l'attribut relationnel `OAE:c3` (lignes 13 et 14). Le concept-rôle $c_{\#3}$ dont l'intension est {`name=account, SC:c6`} est pertinent car il représente une information métier. Par conséquent, même en présense d'une relation circulaire entre $c_{\#3}$ et $c_{\#6}$ ce dernier est jugé pertinent. Enfin, considérons un cas plus complexe de calcul de pertinence, à savoir celui du concept-classificateur $c_{\#7}$ dont les relations avec les autres concepts du treillis sont illustrées par la Figure 6.21. Le concept-classificateur $c_{\#7}$, le concept-rôle $c_{\#2}$ et le concept-association $c_{\#0}$ sont tous des concepts techniques. Ainsi, la pertinence du classificateur $c_{\#7}$ dépend du rôle $c_{\#2}$ qui à son tour dépend de l'association $c_{\#0}$. Une fois que les appels récursifs sont rendus au concept-association $c_{\#0}$, la variable `path` est $\{c_{\#7}, c_{\#2}, c_{\#0}\}$. L'appel de la routine RELEVANT(RLF,$c_{\#2}$,$\{c_{\#7}, c_{\#2}, c_{\#0}\}$) lors du traitement de l'attribut relationnel `FE:c2` au niveau du concept-association $c_{\#0}$ provoque la détection d'un cycle entre le concept-rôle $c_{\#2}$ et le concept-association $c_{\#0}$. Ainsi, l'algorithme décide qu'il est impossible d'évaluer la pertinence du concept-association $c_{\#0}$ (lignes 5 et 6) et par conséquent celle du concept-rôle $c_{\#2}$ et celle du concept-classificateur $c_{\#7}$. Finalement, notons que si un concept est non pertinent tous ses successeurs dans le treillis sont non pertinents aussi.

Propriété 25. *Étant donné un concept c non pertinent. Les concepts appartenant au filtre d'ordre de c, noté* $\uparrow c$*, sont aussi non pertinents.*

Dans un treillis, il y a une relation d'héritage multiple des attributs formels entre un concept et tous ses successeurs (l'ensemble $\uparrow c$). Donc, si un concept est non

pertinent cela signifie que tous ses attributs formels, y compris ceux qu'il a hérité sont techniques et par conséquent tous les successeurs auxquels appartiennent ces attributs hérités sont non pertinents.

6.5.2 Nomage des abstractions

Une fois qu'un concept est jugé pertinent, il est traduit en élément UML et rajouté au modèle UML final. Les éléments qui ont été rajoutés au modèle UML final et qui étaient présents dans le modèle initial ne posent pas un problème de nomage car les mêmes noms peuvent être récupérés à partir de l'attribut formel dont le préfixe est '`name`. Toutefois, les nouvelles abstractions dans le modèle final restent à nommer. Elles requièrent par conséquent un mécanisme de nomage. Plusieus méthodes peuvent être envisagées telles que :

- **Approche manuelle** : le concepteur peut intervenir, a posteriori, pour nommer toute nouvelle abstraction en se basant sur la sémantique qu'elle représente. Ceci est le cas, par exemple, de l'abstraction induite par le concept-classificateur $c_{\#8}$ que nous avons décidé de nommer '`Client`' dans le modèle UML final,
- **Approche lexicale** : on peut aussi procéder par une tokenisation des noms des éléments dérivés de l'abstraction en question et ensuite obtenir le nom de cette abstraction à partir d'une factorisation des termes communs. Dans le cas particulier de la langue Française (Anglaise), un préfixe (suffixe) partagé par les noms d'un ensemble d'éléments peut servir au nomage de l'abstraction qui les représente. Ceci est le cas, par exemple, de l'abstraction induite par le concept-classificateur $c_{\#6}$ qui a été nommé '`Account`' dans le modèle UML final en se servant des noms des deux classes dérivées `MortageAccount` et `CheckAccount`,
- **Approche sémantique** : utilisation des ressources linguistiques telles que des thésaurus, dictionnaires, WordNet[5], EuroWordNet[6], etc. pour la re-

[5]http ://wordnet.princeton.edu/

[6]http ://www.illc.uva.nl/EuroWordNet/

cherche d'un sens commun aux éléments dérivés de l'abstraction en question ; ensuite chercher un nom adéquat pour ce sens.

Pour notre part, nous avons utilisé l'approche sémantique dans la construction de la FCR. En effet, comme nous allons le voir dans le Chapitre 7, Section 7.2, les liens d'homonymie et/ou de synonymie entre les noms des éléments du modèle UML sont étudiés à l'aide du thesaurus électronique WORDNET[7] afin d'établir des regroupements sémantiques entre ces éléments. De plus nous nous sommes appuyés sur l'approche manuelle pour affecter des noms aux nouvelles abstractions de classes, de rôles et d'associations dans le modèles UML final.

6.5.3 Méthode de génération du modèle UML

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 du 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-classificateurs de départ,
- Recherche des concepts pertinents parmis les successeurs des concepts retenus à l'étape précédente,
- Génération du modèle UML par la traduction en éléments UML des concepts-classificateurs retenus et des concepts des autres treillis qu'ils désignent.

6.5.3.1 Sélection des concepts-classificateurs de départ

: cette étape consiste à déterminer les concepts-classificateurs qui correspondent aux classes du modèle initial. Ces concepts sont des concepts-objet dans le treillis final des classificateurs. Le calcul de la SHG permet de réduire l'espace de recherche.

[7] http ://wordnet.princeton.edu/

Ces concepts-objets, une fois traduits en UML, représentent forcément des classes métier. Par exemple, les concepts-objet du treillis final des classificateurs illustrés par la Figure 6.17 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.

6.5.3.2 Recherche des concepts pertinents

: cette étape consiste à parcourir les concepts-classificateurs retenus à l'étape précédente et chercher parmi leurs successeurs des concepts-classificateurs pertinents. Les concepts pertients, rappelons le, sont des concepts dont l'intension contient des informations métiers. Les concepts en relation avec d'autres concepts tirent leur pertinence de celle des concepts désignés. Par exemple, les concepts-classificateurs successeurs des concepts retenus lors de la première étapes sont : $c_{\#6}$, $c_{\#7}$, $c_{\#8}$ et $c_{\#2}$. L'étude de la pertinence de ces concepts-classificateurs donne le résultat suivant :

- $c_{\#6}$ = ({OA:c0, OA:c3, OAE:c3, OO:c0}, ∅) : concept pertinent car il permet de factoriser des éléments de modélisation métier suivants : l'attribut `number` (OA:c0), l'attribut `balance` (OA:c3), le rôle `account` (OAE:c3) et l'opération `credit` (OO:c0).
- $c_{\#8}$ = ({OAE:c0}, ∅) : concept pertinent car il factorise le rôle `owner` (attribut relationnel OAE:c0) ; de plus il introduit une abstraction significative dans le domaine (classe `Client`),
- $c_{\#7}$ = ({OAE:c2}, ∅) : concept non pertinent car le rôle OAE:c2 est une abstraction technique (`visibility=private`, `ordering=unordered`,etc.)
- $c_{\#2}$ = ({`visibility=public`, `isClass`}, {`Double`, `String`}) : concept non pertinent car technique.

À noter que dans le cas de la présence de plusieurs attributs relationnels dont les concepts indiqués sont comparables (labeling non simplifié), il faut considérer l'attribut qui indique le concept le plus spécifique car il dispose d'une plus grande intension et par conséquent plus d'informations sur l'élément de modélisation as-

socié.

6.5.3.3 Génération du modèle UML

: une fois que tous les concepts-classificateurs pertinents sont calculés, la génération des éléments UML consiste à réaliser les traductions suivantes :

1. les concepts-classificateurs retenus sont traduits en classes UML,
2. les liens d'héritage sont déduits de la relation d'ordre entre les concepts-classificateurs,
3. l'intension de chaque concept-classificateur retenu est traduite en éléments de modélisation tels que les attributs, les rôles, les opérations, etc.

6.5.4 Algorithme de génération du modèle UML

L'algorithme 8 code l'ensemble des étapes de génération du modèle UML à partir d'une FTR. L'algorithme commence par le calcul d'un ensemble de concepts-classificateurs candidats de départ en invoquant la routine COMPUTE-STARTING-CONCEPTS (ligne 7) permettant de retourner les concepts-objets du treillis final des classificateurs qui correspondent aux classes du modèle UML initial. Ensuite, l'algorithme parcourt l'ensemble des concepts-classe calculé (lignes 9-17) pour déterminer leurs successeurs pertinents. Finalement, l'algorithme traduit les concepts-classificateurs retenus en UML. L'algorithme utilise plusieurs structures pour stocker les différents types de concepts-classificateurs, à savoir, les concepts qui seront examinés (ensemble `Candidates`), les concepts déjà examinés et qui ont été considérés comme étant pertinents (ensemble `Relevant`), les concepts non pertinents (ensemble `Dummy`) et les concepts dont la pertinence n'a pu être établie (ensemble `Unknown`). Ainsi, au cours du premier parcours (lignes 9-17), la pertinence de chaque candidat est évaluée. Trois résulats se présentent : le concept candidat est pertinent, il est non pertinent ou encore il est impossible de déterminer sa pertinence. Dans le cas où le concept est pertinent (ligne 11), il est rajouté à l'ensemble des concepts-classificateurs qui vont être traduits en classes UML (ligne 12). De

plus, ses successeurs immédiats sont placés parmis les candidats à examiner au future (ligne 13). Les concepts non pertinents (ligne 14) sont ignorés. En outre, en se basant sur la propriété 25, les concepts appartenant aux filtres d'ordre que ces concepts engendrent sont placés dans une structure dans le but de les écartés de la recherche future (ligne 15). Les concepts-classificateurs dont la pertinence n'a pas pu être établie sont rangés dans un ensemble (ligne 17) afin d'être soumis à l'évaluation de l'utilisateur (ligne 18). Enfin, tous les concepts-classificateurs retenus sont traduits en éléments UML par le biais de la routine INTERPRET et utilisés pour mettre à jour le modèle UML final UPDATE (ligne 20).

```
1: Algorithm FTR2UML( In : RLF a family of lattices, Out : 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 ← ∅, Dummy ← ∅, Unknown ← ∅
9: for all c ∈ Candidates do
10:   switch(RELEVANT(RLF,c,∅))
11:   case 1 :  {relevant classifier-concept}
12:         Relevant ← Relevant ∪ {c}
13:         Candidates ← Candidates ∪ Cov^u(c) \ Dummy
14:   case 2 :  {irrelevant classifier-concept}
15:         Dummy ← Dummy ∪ ↑ c
16:   case 3 :  {relevancy can't be determinated}
17:         Unknown ← Unknown ∪ {c}
18: USER-ASSESSMENT(Unknown,Relevant)
19: for all c ∈ Relevant do
20:   UPDATE(M,INTERPRET(FTR,c))
21: return M
```

Algorithm 8: Génération d'un modèle structuré à partir d'une FTR.

Exemple : l'exécution de l'algorithme 8 avec la FTR du modèle de la banque est comme suit : l'ensemble des candidats de départ est composé des concepts-classificateurs $\{c_{\#0}, c_{\#3}, c_{\#4}, c_{\#5}\}$. Ces concepts représentent tous des informations du domaine et sont par conséquent pertinents. Leurs successeurs $\{c_{\#6}, c_{\#8}\}$ représentent aussi des informations métier et sont aussi pertinents. Le seul successeur dont la pertinence n'a pu être établie est le concept-classificateur $c_{\#7}$ qui est relié de manière circulaire avec le concept-rôle $c_{\#2}$. Il est donc placé dans la liste `Unknown` pour être évalué par l'utilisateur. De notre part, nous avons considéré que le concept-classificateur $c_{\#7}$ et le concept-rôle $c_{\#2}$ sont non pertinents car ils ne représentent aucune information métier (voir Section 6.5.1 pour plus de détails sur les critères de la pertinence). Pour terminer, les concepts-classificateurs $c_{\#0}$, $c_{\#3}$, $c_{\#4}$, $c_{\#5}$, $c_{\#6}$ et $c_{\#8}$ sont traduits en classes UML et correspondent au classes `CheckAccount`, `MortageAccount`, `CheckBookHolder`, `Mortgager`, `Account`, `Client`, respectivement.

La Figure 6.22 montre la traduction en éléments UML de tous les concepts-classifcateurs pertinents ainsi que tous les autres types de concepts qui lui sont affiliés. Comparativement au modèle initial de la Figure 6.1, 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.

6.6 Discussion

Nous avons présenté dans ce chapitre la manière d'appliquer le cadre de l'ARC à la construction des hiérarchies de classes. L'ARC prend en compte tous les éléments d'un modèle de classes UML. Pour ce faire, elle s'appuie sur le méta-modèle et associe à chaque classificateur du méta-modèle un contexte dans la FCR. Les associations entre les éléments du méta-modèle servent pour la définition des relations inter-contextes. De cette manière, des abstractions sont obtenues sur tous les types d'éléments du modèle et la factorisation proposée couvre aussi tous ces éléments.

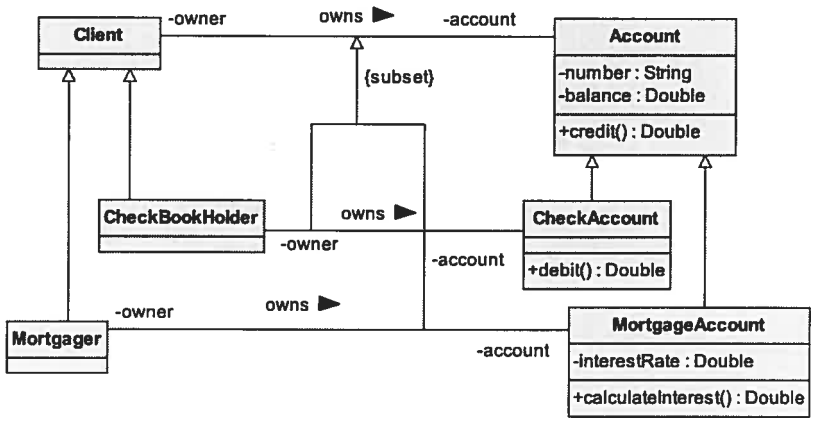


FIG. 6.22 – Le modèle UML final obtenu après la restructuration du modèle initial de la Figure 6.1.

Des recherches complémentaires doivent être conduites permettant la paramétrisation de la méthode del'ARC, notamment pour pouvoir effectuer une factorisation en s'appuyant sur un sous-ensemble d'éléments du méta-modèle ou encore réduire le bruit dans les résultats (les éléments les moins utiles dans le modèle proposé). De plus, il faut introduire plus d'interaction au niveau de l'éxecution de la méthode de l'ARC pour permettre au concepteur d'intervenir au moment du déroulement de la méthode afin de filtrer les éléments indésirables du modèle final. Finalement, d'autre techniques de calcul de similarité et/ou des ressources linguistiques peuvent être expérimentée afin d'accroître la précision dans la résolution des conflits de nommage.

Dans le chapitre suivant nous allons présenter quelques résultats de l'application de la méthode de l'ARC à des diagrammes de classes de milieux industriels.

Chapitre 7

Outils de l'expérimentation

Ce chapitre aborde les outils qui implémentent les méthodes et les mécanismes du cadre de l'ARC. Nous allons, principalement, présenter la plateforme GALICIA qui représente le projet pilote de l'ARC ainsi que l'outil RCFMODELER permettant d'interfacer GALICIA avec les ateliers de génie logiciel les plus courants. Pour terminer, nous expliquerons comment l'outil RCFMODELER effectue quelques opérations de calcul de similarité afin de lever l'ambiguité sur les noms des éléments d'un modèle de classes UML lors de la construction de la RCF associée en s'appuyant sur des ressources linguistiques.

7.1 Galicia

GALICIA [84] est une plateforme logicielle intégrée dédiée à l'AFC permettant d'effectuer les opérations élémentaires de manipulation de treillis, notamment la définition de contextes et la construction/visualisation de treillis ou leurs sous structures dérivées. GALICIA est implémentée en Java[1] et est à code source ouvert distribué sous la licence[2] LGPL.

Au départ, GALICIA était un outil de génération de treillis. Par la suite, ses spécifications ont été étendues pour couvrir plusieurs aspects de la maintenance

[1] http ://www.iro.umontreal.ca/~galicia

[2] http ://www.sourceforge.net/projects/galicia/

des treillis tels que l'assemblage de treillis et le calcul des ordres partiels dérivés. Afin de supporter son utilisation dans le cadre d'applications de *data mining*, des fonctionnalités ont été rajoutées notamment le calcul de différents types de bases de règles d'association [1]. D'autre part, Nous avons utilisé la version 2.0 de la plateforme dans nos expérimentations. Toutefois, le projet GALICIA est en pleine effervescence. En effet, la version 3.0 de la plateforme est déjà sortie et dispose de plusieurs nouveautés tant sur le plan de l'utilisabilité que celui des performances. Le but est de détacher le noyau de GALICIA, des algorithmes utilisés et des opérations d'entrées/sorties. L'Annexe-A décrit la plateforme GALICIA et illustre étape par étape l'utilisation de GALICIA dans le cadre d'une annalyse relationnelles de données en commençant par la composition de la FCR jusqu'à la construction des treillis relationnels en passant par le paramétrage de la méthode MULTI-FCA.

7.2 RcfModeler

L'objectif principal de l'outil RCFMODELER[3] est d'interfacer la plateforme GALICIA avec les outils de modélisation visuelle actuellement disponibles, tels que Rose[4] ou MagicDraw[5]. À cette fin, le langage XMI[6] est utilisé en tant que format d'échange entre, d'une part, les outils UML. et d'autre part, la plateforme GALICIA.

RCFMODELER a été implémenté sous la forme d'un outil séparé de GALICIA afin d'offrir plus de flexibilité dans son développement (par rapport à l'architecture de GALICIA) et sa maintenance à cause de la volatilité des formats de données utilisés dans GALICIA, notamment XMI. RCFMODELER permet donc le codage d'un modèle statique UML exprimé en XMI, version 1.2, en une FCR. À noter qu'un autre outil en cours de développement se chargera de l'opération inverse, c'est-à-dire, la génération d'un modèle UML structuré à partir d'une famille de

[3]Développé par une équipe d'étudiants en maîtrise dans le cadre du cours gradué IFT6310. Site du projet : http ://www-etud.iro.umontreal.ca/ guyomarj/ift6310/projet/index.htm

[4]http ://www-306.ibm.com/software/rational/

[5]http ://www.magicdraw.com/

[6]http ://www.omg.org/technology/documents/formal/xmi.htm

treillis produite par GALICIA.

RCFMODELER crée un contexte binaire pour chacun des types d'éléments du diagramme de classes suivants : les classificateurs, les attributs, les opérations, les associations et les rôles. Ensuite, dépendamment de l'élément du diagramme de classes, RCFMODELER permet de préciser les propriétés à considérer lors du codage en une RCF de ce diagramme telles que la visibilité, la multiplicité, la portée, etc. Ces propriétés deviendront des attributs formels dans le contexte qui correspond au type d'éléments décrits.

Comme RCFMODELER manipule des modèles de classes UML, il faut s'attendre à des conflits de noms entre les éléments du modèle (pour plus de détails, voir Chapitre 8.2.2.2, Section 6.3). Pour cela, il fait appel à des outils linguistiques afin de détecter les situations suspectes dans un modèle où des conflits potentiels de noms résident. Ainsi, l'outil permet de rechercher de possibles regroupements sémantiques entre les éléments du modèle en exploitant les noms utilisés pour les désigner. À cette fin, les liens d'homonymie et/ou de synonymie entre ces noms sont étudiés à l'aide du thésaurus électronique WORDNET[7].

Dans la suite de la section, nous allons décrire les outils utlisés par RCFMODELER, notamment le thesaurus WORDNET et le moteur de recherche LUCENE ainsi que les mécanismes implémentés pour le calcul de la similarité entre les noms des éléments du diagramme de classes UML.

7.2.1 WordNet

WORDNET [24] est un dictionnaire sémantique de l'anglais accessible depuis un site Web et une interface de programmation (API). Il représente une ressource lexicale libre usage très utilisée de nos jours en matière de traitement de langages naturels en général et la désambigüisation sémantique en particulier. WORDNET organise l'information lexicologique en termes de signification de mots plutôt que le mot en tant que forme. Son fonctionnement se base sur un système de relations

[7]http ://wordnet.princeton.edu/

entre les mots telles que synonymie, méronymie, polysémie, hyponymie, antonymie, etc. La synonymie est la relation sémantique fondamentale qui permet de définir les mots. Ainsi, pour un mot donné, WORDNET retourne la définition, les synonymes (synset), les homonymes, les formes dérivées et des fragments de phrases. Pour d'autres langues telles que le Français, il existe d'autres ressources notamment EUROWORDNET[8]. Toutefois, WORDNET est maintenant un outil universel, reconnu et gratuit.

7.2.2 Lucene

LUCENE[9] est un moteur de recherche en code source ouvert très flexible. Il s'intègre directement à une application donnée (ici RCFMODELER) car il est implémenté en Java par le groupe Jakarta Apache et son utilisation repose entièrement sur des API. L'application peut utiliser LUCENE comme noyau pour toute fonctionnalité de recherche d'information. Les données de l'application sont, d'abord, utilisées pour créer une collection de documents. Les documents sont ensuite soumis à LUCENE pour être analysés, indexés et stockés dans son système de fichiers grâce à son rédacteur d'index.

L'application peut composer une requête et demander à LUCENE de l'analyser et d'effectuer la recherche dans son index. LUCENE retourne la collection des documents qui correspondent à la requête. Les documents de la collection sont triés par ordre de pertinence. La pertinence d'un document est obtenue par le mécanisme de scoring de LUCENE. Le moteur de recherche LUCENE repose sur le modèle vectoriel et détermine la distance du cosinus[10] dans le calcul de la similarité. Rappelons que selon ce modèle, la distance entre la requête et le document correspond au cosinus de l'angle formé par les vecteurs respectifs dans un espace multi-dimentionnel.

[8] http ://www.illc.uva.nl/EuroWordNet/

[9] http ://lucene.apache.org/java/docs/index.html

[10] http ://lucene.apache.org/java/docs/api/org/apache/lucene/search/Similarity.html

7.2.3 Mesure de similarité

Pour résoudre les conflits de noms entre les éléments du modèle UML, RCF-MODELER utilise deux méthodes pour le calcul de similarité entre les noms du modèle UML, l'une s'appuie sur WORDNET et l'autre est basée sur des techniques de recherche d'information en utilisant le moteur de recherche LUCENE.

Avant de se lancer dans toute recherche de similarité entre les noms du modèle, RCFMODELER prépare ses données. En effet, au moment du chargement du fichier XMI qui correspond au modèle UML, l'analyseur[11] de RCFMODELER crée ses propres index, un par type d'élément UML (classe, attribut, méthode, etc.). À chaque nom d'élément est associé une entrée dans l'index approprié. La valeur associée à cette entrée dans l'index est calculée de la manière suivante : d'abord l'analyseur procède à la tokenisation du nom de l'élément, par exemple le nom de classe `CheckBook` est séparé en `Check` et `Book` ; ensuite, toutes les lettres sont traduites en minuscule, dans notre exemple `check book`. Enfin, RCFMODELER extrait les sens possibles des différents termes 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 (la clé dans l'index) et d'autre part de ses termes avec leurs synsets (la valeur associée à la clé) est placé dans l'index approprié. Une fois que les index sont créés, ils peuvent être exploités pour calculer la similarité des noms des éléments du modèle UML. Ce calcul peut être réalisé de deux manières différentes :

7.2.3.1 Distance WordNet

Pour trouver les noms d'éléments UML similaires, RCFMODELER parcourt les noms des éléments de même type (attributs, méthodes, etc.) et calcule les distances deux à deux. La mesure de similarité entre deux noms repose sur un appariement optimal[12], initialement implémenté dans l'outil d'alignement d'ontologies OLA[13]

[11] Le module RCFMODELER.LING.

[12] Best-matching

[13] http ://www.iro.umontreal.ca/~owlola/align_files.html

(OWL-Lite Alignment) [44] et dont le principe est le suivant.

Étant donné deux termes composites dont on veut mesurer la distance, le principe de la méthode consiste d'abord à représenter ces deux termes par un graphe biparti dans lequel les sommets sont répartis en deux pools. Les sommets de chaque pool représentent les termes atomiques de l'un des noms composites. Les arêtes dans ce graphe ont une extrémité dans chaque pool et correspondent aux différentes possibilités d'appariement entre les termes des deux noms. Un poids est affecté à chaque arête. Ce poids est égal à la distance sémantique [11] dans l'arbre de WORDNET en considérant les deux termes atomiques en tant que noms, verbes et adjectifs et en prenant le maximum. La Figure 7.1 illustre le graphe qui correspond aux noms `invoiceAmount` et `customerOrderSum` avec les distances WORDNET entre les termes. Par exemple, les distances entre le terme `invoice` et `order` en tant que noms est égale à 0.84 tandis que la distance en tant que verbes et adjectifs est nulle.

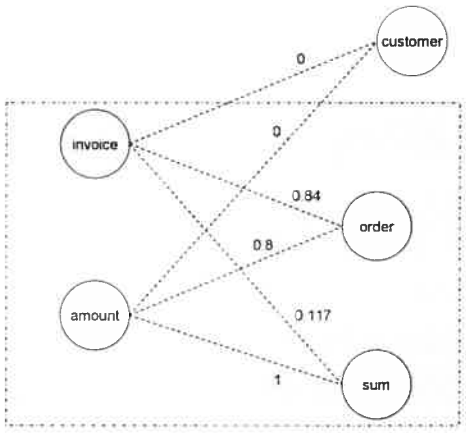


FIG. 7.1 – Graphe biparti qui correspond aux noms `invoiceAmount` et `customerOrderSum` avec les distances WORDNET entre les termes.

Pour mesurer la similarité entre deux noms, la méthode cherche un appariement de taille maximale[14] et exclusif entre les pools de termes respectifs. Afin d'obtenir un appariement de taille maximale, la méthode sature le plus petit pool de noeuds

[14]En nombre de sommets.

en selectionnant à chaque fois les sommets dont les arêtes associées possèdent le plus grand poid. Par exemple, le sommet `amount` est relié aux sommets `sum` et `order` avec les poids 1 et 0.8, respectivement. Les deux sommets `amount` et `sum` sont donc retenus car la similarité est normalisée dans [0,1]. Un appariement exclusif signifie que les arêtes ne partagent pas les mêmes sommets. Par exemple, dès que les deux sommets `amount` et `sum` ont été selectionnés, ils sont écartés pour garantir l'exlusivité et `invoice` est par la suite comparé à `order` et `customer` afin d'obtenir la saturation. La distance globale entre les deux mots est la somme des poids des arêtes de l'appariement retenu divisée par la taille du pool le plus large du graphe [23]. Dans le cas des deux mots `invoiceAmount` et `customerOrderSum`, l'appariement retenu est composé des pairs (`amount`,`sum`) et (`invoice`,`order`). Par conséquent, la mesure de similarité totale est (1+0.84)/3=0.614.

7.2.3.2 Techniques de la recherche d'information

D'abord, les données de RCFMODELER (les noms des éléments du modèle UML avec leurs synsets) sont utilisées pour créer la collection de documents du moteur de recherche LUCENE. Le nom de l'élément avec ses synsets est considéré comme un document de cette collection. Les documents sont indexés par LUCENE. Ensuite, chaque nom d'élément est considéré comme une requête dans le moteur de recherche LUCENE. La requête est composée du nom de l'élément, de ses termes ainsi que le synset de chaque terme. 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 éléments de la collection. Il retourne une liste des noms triés selon le degré de similarité.

7.2.4 Production des noms communs

Les deux méthodes de calcul de similarité retournent une liste de noms triés par ordre de similarité. RCFMODELER se sert du seuil de similarité (paramètre Min-relevancy) fixé au préalable par l'utilisateur pour selectionner les éléments les plus similaires et les considère comme synonymes après approbation de l'utilisateur.

Ainsi, pour un couple de mots dont la distance est supérieure au seuil spécifié, l'outil RCFMODELER demande à l'utilisateur de valider la similarité. Ceci se fait dans le but d'éviter de se fier seulement à l'identation des éléments du modèle et d'ignorer la sémantique associée. Pour une similarité valide, l'outil demande à l'utilisateur s'il veut attribuer un nom commun au groupe de noms initiaux, dit 'noms de base'. Sinon, l'outil utilise les noms de base pour créer un nom commun qui sera utilisé lors de la génération des contextes de la RCF. Ce nom est obtenu par la concaténation des noms de base afin de garder une trace de l'opération. De plus, les termes communs des noms de base sont factorisés.

Durant la phase de génération de la FCR associée au modèle UML chargé, RCFMODELER agit de deux manières différentes, selon le degré de similarité entre noms (voir Chapitre 8.2.2.2, Section 6.2.2 pour plus de détails sur le codage des attributs) :

- Si les noms sont identiques, RCFMODELER produit un seul attribut formel `name='nom-commun'` au lieu de produire autant d'attributs formels de type `name='nom-de-base'` qu'il y a de noms similaires. Par exemple, les deux attributs de classe **sum** et **amount**, qui sont traduits en deux objets formels dans le contexte des attributs de classe, sont supposés avoir l'attribut formel **name=sum** et **name=amount**, respectivement. Toutefois, les deux noms sont identiques (voir la distance qui les sépare sur la Figure 7.1). En utilisant le nom commun **amount** par exemple, les deux objets formels sus-cités vont devoir partager l'attribut formel commun **name=amount** tandis que les deux attributs formels **name=amount** et **name=sum** vont disparaitre du contexte.
- Si les noms ne sont pas identiques mais seulement similaires, l'outil code les

objets formels associés aux éléments ayant des noms similaires comme s'il n'y avait pas de similarité de nom, c'est-à-dire, en produisant pour chacun un attribut formel de type `name='nom-de-base'` ; ensuite produit un attribut formel de la forme `name='nom-commun'` qui sera partagé par tous les objets formels ayant des noms similaires.

Paramétrage de RcfModeler : Il faut préciser à RCFMODELER la méthode à utiliser pour calculer la similarité. Le choix est entre le calcul de la distance WORDNET ou la recherche en utilisant LUCENE. Dans le cas d'une recherche avec LUCENE, il faut préciser s'il faut faire une recherche lisse ou non par rapport à la fréquence des termes dans l'index. Une recherche lisse consiste à prendre en charge les termes très fréquents lors du calcul de la similarité et à les considérer comme étant moins pertinents par rapport a la fréquence des termes dans l'index. Par exemple, la présence dans deux noms d'éléments des termes `get` et `set`, qui s'avèrent très fréquents dans l'appellation des méthodes de classes, serait normalement moins pertinent étant donné que l'index des méthodes en contiendrait beaucoup. Une recherche non lisse consiste à utiliser le calcul de la distance du cosinus avec tf-idf sans aucune restriction sur la fréquence des mots dits 'vides de sens' tels que `get` et `set`.

D'autre part, les deux paramètres Min relevancy et Max index proportion utilisés par les algorithmes de calcul de similarité sont un moyen pour réduire la quantité d'information retourné à GALICIA. En effet, Min relevancy est la valeur minimale de pertinence à considérer lors de la selection des documents obtenus par le calcul de similarité et Max index proportion est la proportion de synonymes à retenir pour un terme d'un nom d'élément par rapport à la taille de l'index respectif. Par exemple, on peut décider que le nombre des synonymes ne doit pas dépasser la moitié de la taille de l'index. Ceci afin d'éviter que le nombre élevé de synonymes de certains termes ne déroute le véritable sens d'un nom d'élément. Finalement, nous signalons que les valeurs de ces paramètres sont normalisées pour se ranger entre 0 et 1.

7.3 Conclusion

Ce qui distingue la plateforme GALICIA des autres outils de manipulation de contextes/treillis développés par la communauté FCA est la possibilité d'analyser les données relationnelles grâce au cadre de l'ARC qui y est implémenté. De plus, les outils qui gravitent tout au tour, notamment RCFMODELER font de GALICIA un moyen d'aide à la restructuration des diagrammes de classes UML comme nous allons le voir dans le Chapitre 8. Toutefois, des efforts de développement supplémentaires doivent être réalisés afin d'améliorer les aspects de visualisation de treillis, de navigation dans les treillis et d'interfaçage avec les ateliers de génie logiciel usuels. Ce dernier aspect doit être profondément reinvesti notamment pour le support des différentes versions du format XMI que les outils de modélisation UML utilisent.

Chapitre 8

Projets d'expérimentation de l'ARC

L'ARC a été appliquée à la restructuration des modèles statiques de UML (diagrammes de classes) dans le cadre du projet MACAO[1]. Nous allons présenter, dans ce chapitre, les différents projets de validation de l'ARC comme une technique de construction/restructuration de modèles.

8.1 Macao

L'application de l'ARC à la restructuration des modèles statiques UML a été étudiée et validée dans le cadre du projet MACAO[2] (Modélisation et Audit de Composants A Objets). Dans le cadre de ce projet, divers systèmes propriétaires de France Télécom ont été analysés. 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 télecommunication. Les modèles de classes correspondants sont de niveau analyse et conception. Pour des

[1]Un projet réalisé conjointement avec France Télécom, SofTeam et le LIRMM, et supporté par le Reseau National de recherche et d'innovation en Technologies logicielles (RNTL), France.

[2]Un projet entre France Télécom, SOFTEAM et LIRMM, financé par le réseau national de recherche et d'innovation en technologies logicielles (RNTL), France. http ://www.lirmm.fr/macao.

raisons de confidentialité sur certaines données de France Télécom, nous ne pouvons décrire ces données et nous nous contenterons de préciser l'utilisation du cadre de l'ARC par l'intermédiaire de la plateforme GALICIA.

La plateforme GALICIA a été interfacée avec l'atelier OBJECTEERING[3] permettant ainsi l'utilisation de l'algorithme ICG -une implémentation de la méthode MULTI-FCA avec les SHG- avec les modèle UML issus de OBJECTEERING. Ainsi pour un modèle donné, la FCR est exportée[4] à GALICIA qui se charge par le bais de ICG de produire une famille de SHG, une par contexte de la FCR. Les concepts des différentes SHG sont par la suite traduits en éléments de modèles UML et communiqués de nouveau à OBJECTEERING dans le but de créer un modèle de classes amélioré.

L'application de ICG a permet la découverte de plusieurs factorisations non triviales de classes mais aussi d'associations jugées potentielles par les analystes et les concepteurs. Toutefois, l'information fournie par ICG (nouvelles classes, associations, etc.) est parfois difficile à analyser du fait de la complexité des structures conceptuelles à partir desquelles le modèle de classes amélioré a été obtenu. Ainsi, un ensemble d'outils on été développé dans le cadre du projet MACAO permettant d'assister le concepteur du diagramme de classes dans l'exploitation des résultats de ICG par l'analyse de la factorisation des attributs, méthodes, associations, etc.

8.2 Jetsgo

8.2.1 Jetsmiles

Jetsgo[5] est une compagnie privée de transport aérien du Canada, basée à Montréal, à bas coûts. Elle a été fondée en 2002 et a connu une croissance fulgurante. Elle a commencé ses vols avec trois avions et sept destinations. Elle exploite, deux ans plus tard, 21 avions et dessert 27 destinations dont 15 au Canada et 12

[3]http ://www.objecteering.com

[4]OBJECTEERING offre une paramétrisation limitée pendant la construction de la FCR.

[5]http ://www.jetsgo.net/

aux États-Unis.

Lors de son exercice financier 2004, terminé à la fin de juin, l'entreprise montréalaise qui compte 1200 employés a affiché le double des revenus de l'année précédente. Par la suite, la hausse des prix du carburant et la guerre des prix des billets des compagnies aériennes à rabais ont pesé sur sa santé financière l'obligéant de cesser toutes ses activités et de se mettre sous la protection de la loi sur la faillite jusqu'au 13 mai 2005.

En septembre 2003, Jetsgo a lancé le programme de récompense Jetsmiles qui permet aux grands voyageurs de Jetsgo de cumuler des points pour obtenir des vols gratuits. Ainsi chaque dollar du prix du billet acheté chez la compagnie ou chaque dollar porté à la carte *MasterCard Or Jetsgo*[6] rapporte un Jetsmile de plus pour un voyage gratuit.

La réalisation technique, artistique et promotionnelle de ce programme a été attribué à l'agence BRAQUE. BRAQUE[7] est une agence de création publicitaire montréalaise qui a été crée en 1992 et où l'on offre divers services tels que la planification stratégique en technologie de l'information, la promotion, les relations publiques, design graphique, interactif, etc. L'agence BRAQUE emploi une vingtaine de personne et compte parmi ses clients Vidéotron Télécom Ltée, Bombardier Produits récréatifs, Jetsgo, etc.

Le programme Jetsmiles comprend l'enregistrement des passagers, des itérinaires de voyage ainsi que les points de récompense cumulés. La Figure 8.1 montre le modèle de classes du domaine.

Nous allons nous servir de l'ARC par le biais de la plateforme GALICIA pour détecter de probables anomalies de conception du modèle de classes de la Figure 8.1 et proposer dans le cas échéant un modèle de classes restructuré. Le but de la restructuration est d'augmenter la factorisation des éléments de modélisation (attributs, méthodes, rôles, etc.) et éventuellement l'introduction de nouveaux concepts du domaine. Toutefois, pour des raisons de taille de la FCR qui encode le modèle

[6]En partenariat avec la Banque Nationale du Canada.

[7]http ://www.agencebraque.com/fr/flash.htm

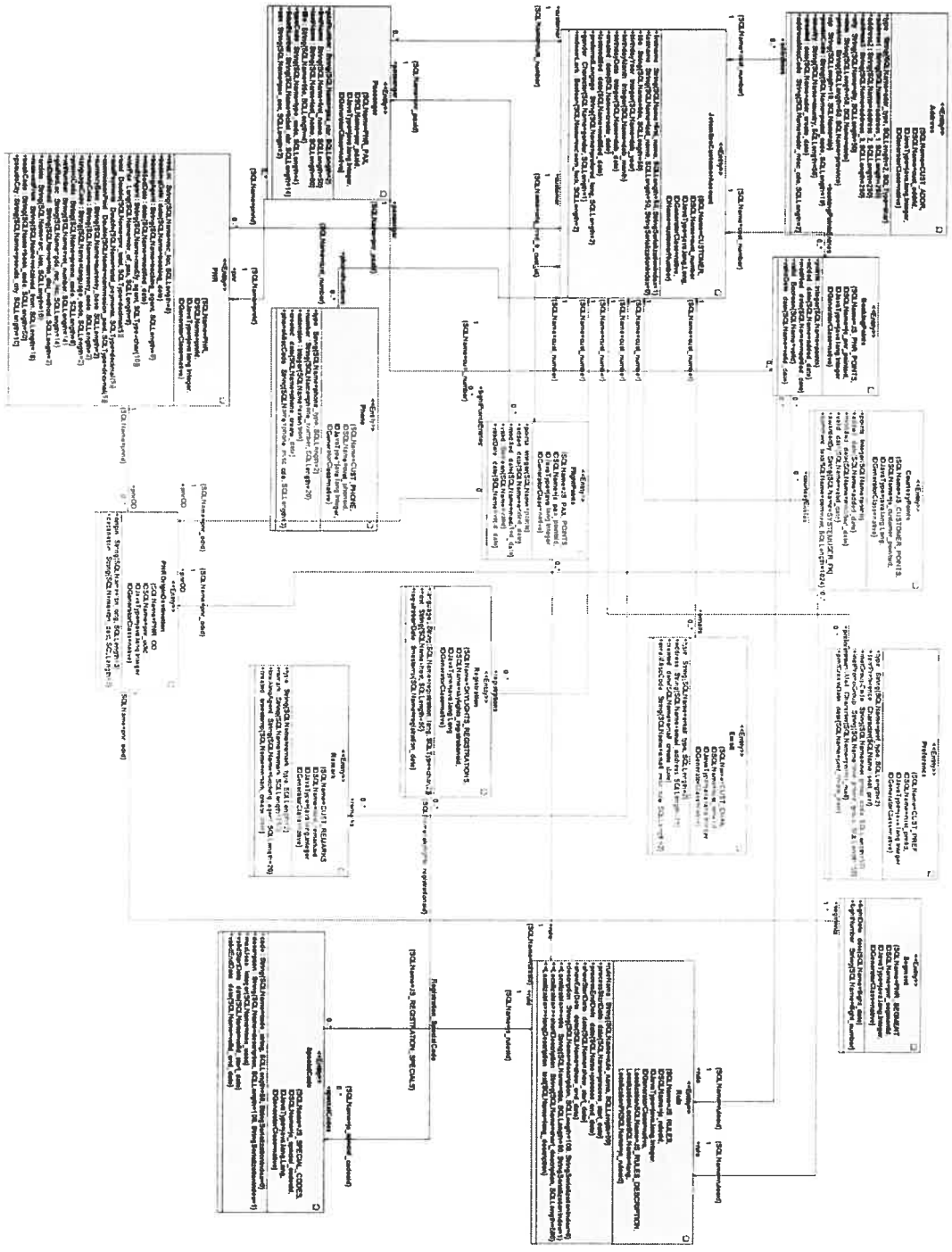


FIG. 8.1 – Modèle du domaine de Jetsmiles.

et des treillis correspondants, nous allons nous limiter au sous ensemble de classes de la Figure 8.2 auquel nous avons ajouté quelques précisions supplémentaires notamment les noms des associations.

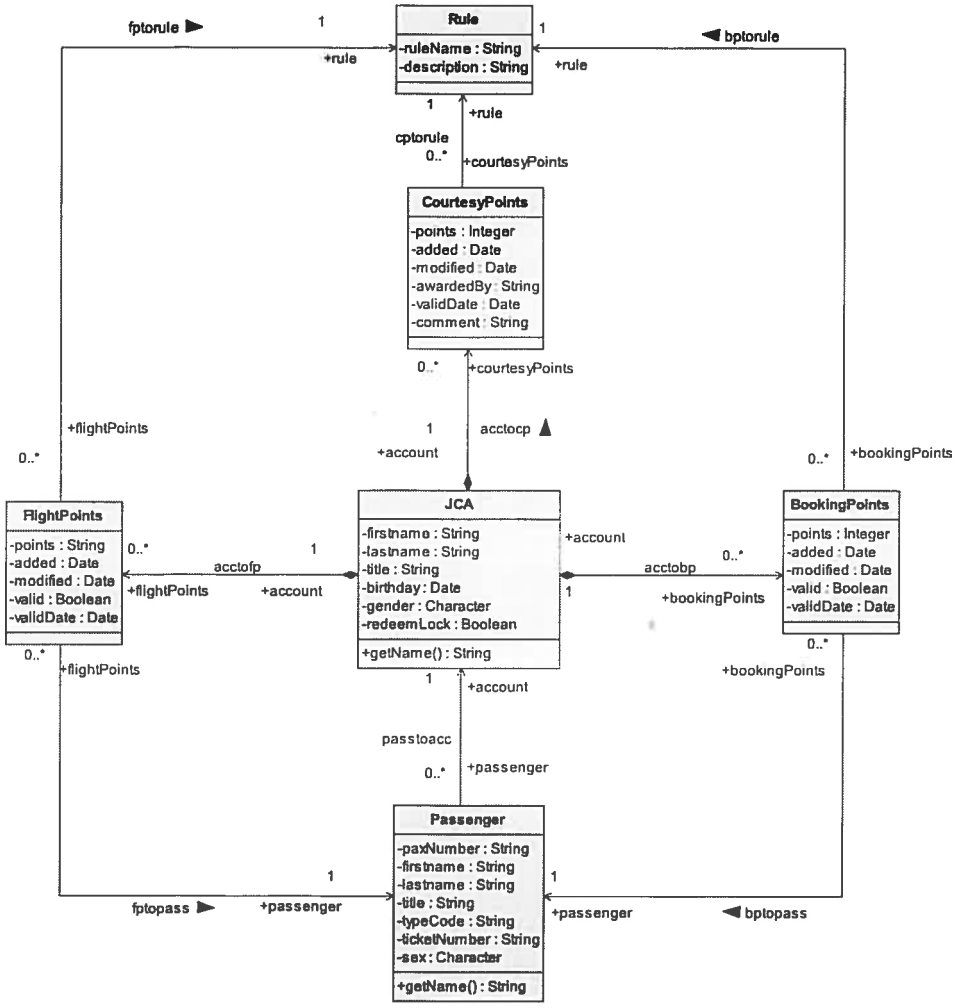


FIG. 8.2 – Sous modèle du domaine de Jetsmiles.

8.2.2 Génération du modèle de classes restructuré

Le modèle de classe de la Figure 8.2 a été réalisé avec l'outil MAGICDRAW 9.5. Il a été ensuite exporté en format XMI 1.2. L'outil RCFMODELER a été finalement utilisé pour produire la FCR décrite en Annexe-C.

La méthode de génération d'un modèle UML (Algorithme 8) consiste à chercher dans le treillis final des classificateurs, illustré par la Figure 8.3, les concepts-classificateurs pertinents (voir Propriété 25) qui vont être traduits en classes. Par exemple, le concept $c_{\#15}$=(`OA :c12`, `OA :c13`, `OA :c14`, `OA :c16`, `OAE :c4`, `OAE :c58`, `OAE :c71`) est traduit en une classe, appelée `Points` dans le modèle de classes final illustré par la Figure 8.8 dont les attributs sont : `points` (`OA :c12`), `added` (`OA :c13`), `modified` (`OA :c14`) et `validDate` (`OA :c16`) et les rôles sont : `AccToPoints.points` (`OAE :c4`) et `PointsToRule.points` (`OA :c71`). On verra par la suite pourquoi l'attribut-rôle `OAE :c58` ne peut être considéré. La génération du modèle UML final à partir de la FTR du modèle JETSMILES est présenté ci-dessous. Nous allons revenir sur certains aspects de la méthode de génération décrite dans le Chapitre 6, section 6.5.3 avec plus d'illustrations, notamment l'étude de la pertinence d'un concept.

8.2.2.1 Sélection des concepts-classificateurs de départ

Les classes du modèle UML initial de la Figure 8.2 correspondent à des concept-objets dans le treillis des classificateurs de la Figure 8.3 et doivent être présents dans le modèle final. Les concepts-objet représentant les classes du modèle de la Figure 8.2 sont : $c_{\#0}$, $c_{\#3}$, $c_{\#4}$, $c_{\#5}$, $c_{\#6}$, $c_{\#7}$ et correspondent aux classes `JCA`, `Passenger`, `BookingPoints`, `CourtesyPoints`, `FlightPoints` et `Rule`, respectivement. Ces concepts sont donc traduits en classes dans le modèle de départ (Figure 8.8) car ils représentent tous des classes métiers. Il reste maintenant à déterminer les autres concepts pertinents et les traduire en généralisations de classes qui factorisent les différentes propriétés en se servant principalement du treillis des classificateurs et des treillis qui lui sont rattachés. Par exemple, on pourrait avoir

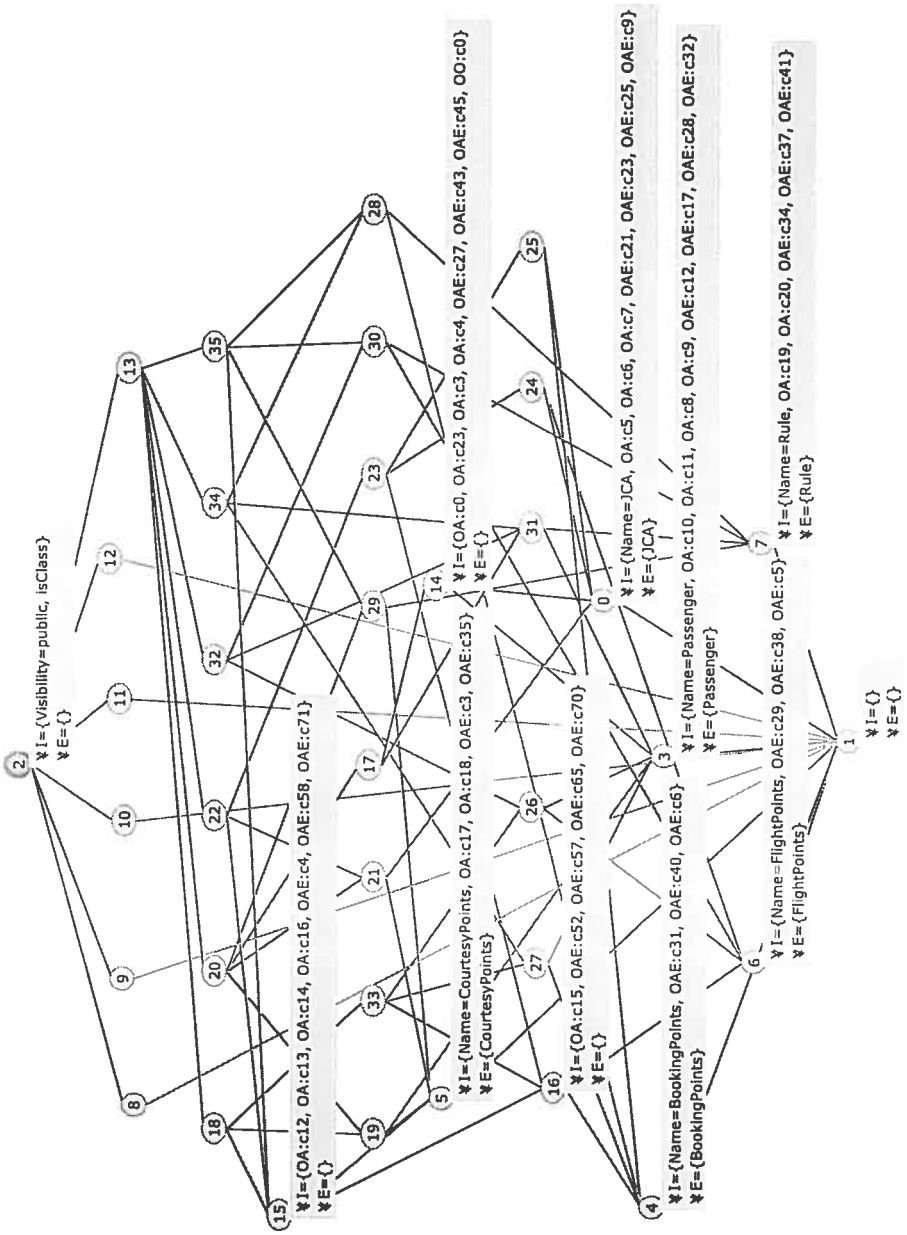


FIG. 8.3 – Treillis des classificateurs du diagramme de classes de la Figure 8.2.

comme seule généralisation des deux classes `BookingPoints` et `FlighPoints` qui correspondent aux concepts-classificateurs $c_{\#4}$ et ($c_{\#6}$) de la Figure 8.3, la classe qui correspond au concept-classificateur $c_{\#15}$. Le concept-classificateur $c_{\#16}$ peut aussi être traduit dans le modèle UML final en une classe généralisant ces deux classes. De ce fait, la génération du modèle UML final est une tâche exploratoire et semi-automatique qui demande des outils d'analyse de treillis et l'intervention d'un concepteur. Les outils permettent la visualisation et la navigation entre treillis suivant les relation inter-concepts ainsi qu'une recherche multi-critèes de concepts. Le concepteur prend les décisions de modélisation appropriées dans le cas où plusieurs alternatives seraient offertes.

8.2.2.2 Recherche des concepts pertinents

À ce stade, il faut déterminer les concepts-classificateurs, appartenant à la couverture supérieure des concepts-classificateurs retenus dans l'étape précédente, qui peuvent être traduits en classes. Cependant, à cause du nombre élevé des concepts du treillis des classificateurs et des abstractions non significatives qu'ils peuvent engendrer au niveau du modèle UML final exigant la mise en place d'un filtre de pertinence (voir Chapitre , Section 6.5.1). Par exemple le concept-classificateur $c_{\#13}$=(`OA :c2`, `OAE :c61`, `OAE :c8`, `visibility=public`, `isClass`) de la Figure 8.3 renferme dans son extension toutes les classes 'public' du modèle de la Figure 8.2. La traduction de ce concept dans le modèle UML final serait une classe non intéressante car elle représenterait toutes les classes du modèle initial dont la visibilité est public. La génération du modèle final de classes consiste donc à explorer le treillis des classificateurs et à choisir les concepts pertinents qui devraient être traduits en classes. Prenons le cas du concept-classificateur $c_{\#16}$=({`OA :c15`, `OAE :c52`, `OAE :c65`, `OAE :c70`},∅). L'interprétation des propriétés de ce concept est comme suit :

- `OA :c15` : un attribut dont les caractéristiques sont données par le concept-attribut $c_{\#15}$ du treillis des attributs illustré par la Figure 8.4, à savoir : `Name=valid`, `Multiplicity=[1,1]`, `InitialValue=null`, `Changeability=`

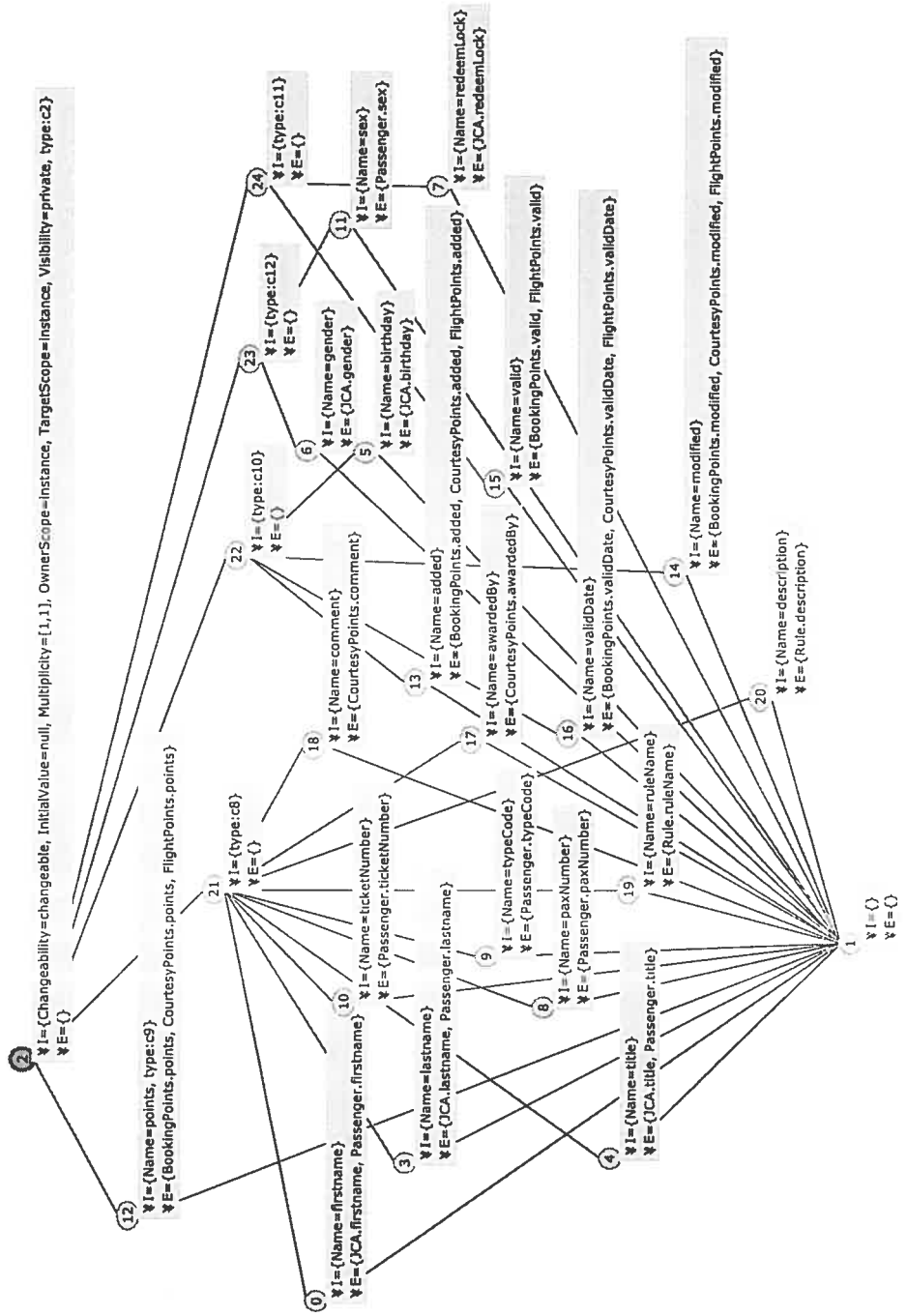


FIG. 8.4 – Treillis des attributs du diagramme de classes de la Figure 8.2.

changeable, OwnerScope=instance, TargetScope=instance, visibility= private, type :c11 (correspond au type Boolean dans le treillis des classificateurs).

- OAE :c70 : un rôle d'association dont les caractéristiques sont illustrées par l'intension du concept-rôle $c_{\#70}$ du treillis final des rôles (treillis trop volumineux pour être donné par une figure) et qui sont : Aggregation=none, BTA :c16, Changeability= changeable, Multiplicity =[0,N], Ordering =unordered, SC :c16, TargetScope=instance et visibility= public. L'attribut association BTA :c16 précise que ce rôle appartient à l'association décrite par le concept-association $c_{\#16}$ du treillis des associations illustré par la Figure 8.5. Le concept-association $c_{\#16}$ dont l'intension est {first_end :c11, second_end :c20} représente une association avec deux extrémités décrites par les concepts-rôles $c_{\#11}$ et $c_{\#20}$. D'abord, comme le concept-rôle $c_{\#11}$ est un super-concept de $c_{\#70}$, alors $c_{\#70}$ est considérée comme étant l'une de ces deux extrémités. Ensuite, le concept-rôle $c_{\#20}$ indique que ce concept représente un rôle de la classe qui correspond au concept-classificateur $c_{\#7}$, à savoir la classe Rule.
- OAE :c52 : un rôle d'association dont les caractéristiques sont illustrées par l'intension du concept-rôle $c_{\#52}$ du treillis final : Aggregation=none, BTA :c11, Changeability=changeable, Multiplicity =[0,N], Ordering=unordered, SC :c16, TargetScope=instance, visibility =public et isNavigable. L'attribut association BTA :c11 précise que ce rôle appartient à l'association décrite par le concept-association $c_{\#11}$ du treillis des associations illustré par la Figure 8.5.
- OAE :c65 : un rôle d'association dont les caractéristiques sont illustrées par l'intension du concept-rôle $c_{\#65}$ du treillis final : Aggregation=none, BTA :c15, Changeability=changeable, Multiplicity =[0,N], Ordering=unordered, SC :c16, TargetScope=instance et visibility =public. L'attribut-association BTA :c15 précise que ce rôle appartient à l'association décrite par le concept-association $c_{\#15}$ du treillis illustré par la Figure 8.5.

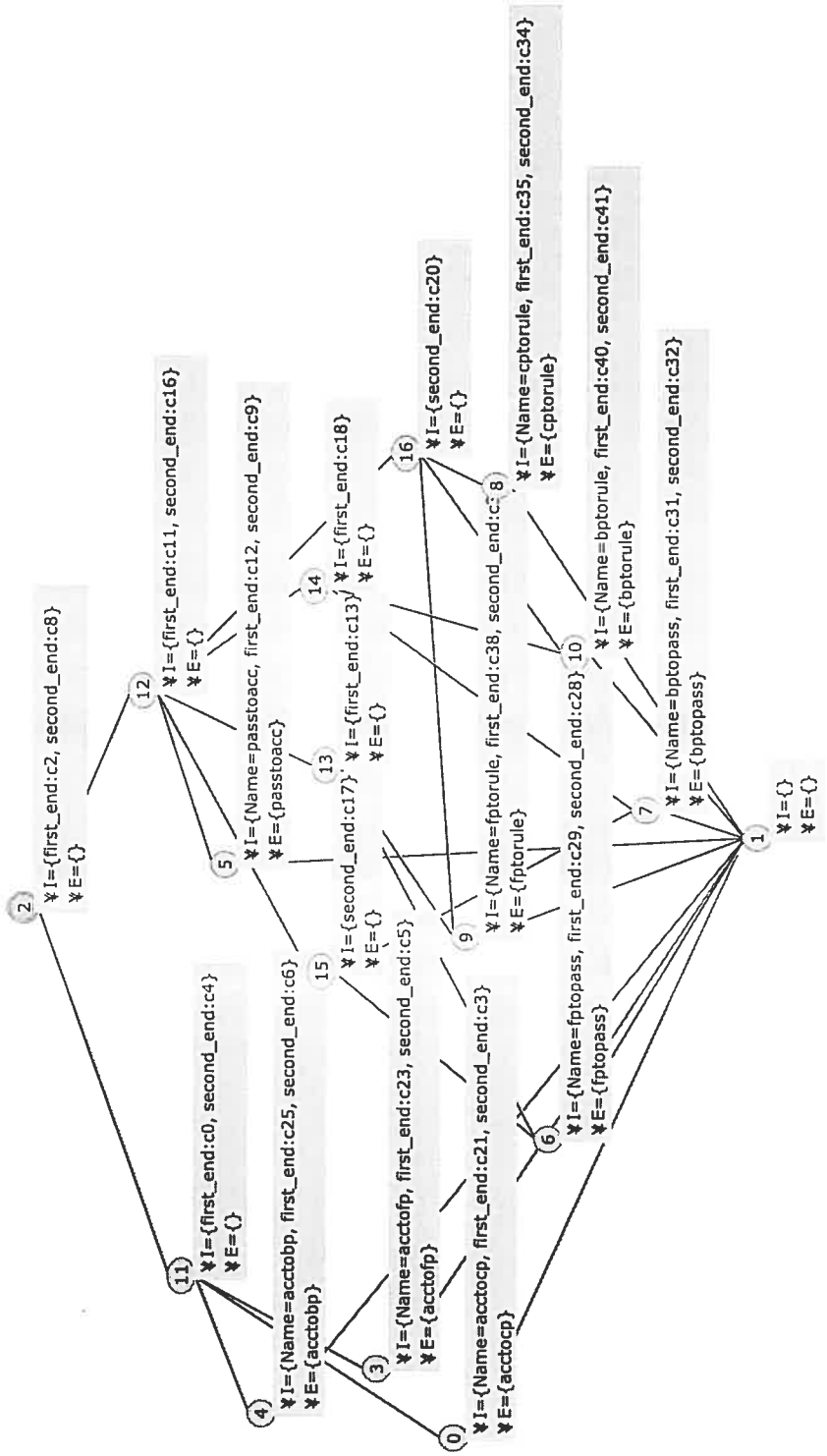


FIG. 8.5 – Treillis des associations du diagramme de classes de la Figure 8.2.

En considérant les concepts-classificateurs $c_{\#0}$, $c_{\#3}$, $c_{\#4}$, $c_{\#5}$, $c_{\#6}$ et $c_{\#7}$, retenus lors de la première étape, nous avons détecté les nouveaux concepts-classificateurs suivants : $c_{\#14}$, $c_{\#15}$ et $c_{\#16}$. En effet, le concept $c_{\#14}$ représente une classe, appelée `JCANPassenger`, regroupant les propriétés partagées par les deux classes de base `JCA` ($c_{\#0}$) et `Passenger` ($c_{\#3}$). Le concept $c_{\#15}$ est traduit en une classe ayant comme propriétés les attributs communs aux classes `BookingPoints` ($c_{\#4}$), `CourtesyPoints` ($c_{\#5}$) et `FlightPoints` ($c_{\#6}$). Les attributs relationnels indiquant des concepts non retenus ne sont pas considérés. Ceci est le cas, par exemple, de l'attribut-rôle `OAE :c45` appartenant au concept-classificateur $c_{\#14}$ qui désigne le concept-rôle $c_{\#45}$ dont l'intension est : `BTA :c2, Changeability=changeable, Multiplicity=[1,1], Ordering=unordered, SC :c14, TargetScope=instance` et `visibility=public`. Le concept-rôle $c_{\#45}$ appartient donc à l'association décrite par le concept-association $c_{\#2}$ dont l'intension est `first_end :c2, second_end :c8`. L'attribut-rôle `second_end :c8` dont l'intension désigne un concept-classificateur qui n'a pas été retenu pour le modèle UML final. Par conséquent, l'association déduite à partir du concept-association $c_{\#2}$ est ignorée ainsi que les rôles qui lui sont affiliés notamment celui représenté par l'attribut-rôle `OAE :c45` dans le concept-classificateur $c_{\#14}$.

8.2.2.3 Génération du modèle UML

Les concepts-classificateurs retenus sont traduits en classes UML. Les liens d'héritage sont déduits de la relation d'ordre entre ces concepts-classificateurs. Ensuite, pour chaque concept-classificateur, son intension est examinée pour être traduite en éléments de modélisation tels que attributs, rôles, opérations, etc. Ces opérations sont réalisées par la primitive INTERPRET de l'algorithme 8. La Figure 8.6 illustre la portion du modèle final obtenue par l'analyse du concept-classificateur $c_{\#16}$.

La Figure 8.7 montre la correspondance entre les éléments du diagramme de classes final et les concepts appartenant aux différents treillis.

La Figure 8.8 montre le diagramme de classe final où les nouvelles classes,

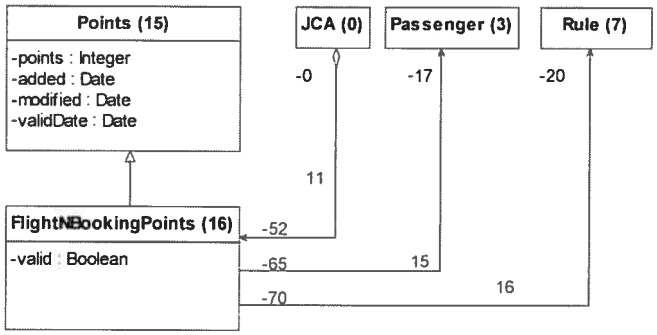


FIG. 8.6 – Portion du model UML final obtenue après le traitement du concept $c_{\#16}$ du treillis des classificateurs.

rôles et associations ont été renommés. On constate dans cette version améliorée du modèle de classes de l'application `Jetsmiles` une meilleur factorisation des attributs et des rôles. Trois nouvelles classes ainsi que des associations qui vont de pair ont été rajoutées.

8.3 Conclusion

Nous avons présenté dans ce chapitre quelques résultats de l'application de la méthode de l'ARC à la construction des hiérarchies de classes avec des diagrammes de classes dans les milieux industriels. Dans le cadre de ces projets de validation, les concepteurs des hiérarchies de classes ont fait la découverte de plusieurs classes pertinentes parmi celles qui ont été suggérées. Cependant, des outils de navigation de treillis se sont avérés très utiles.

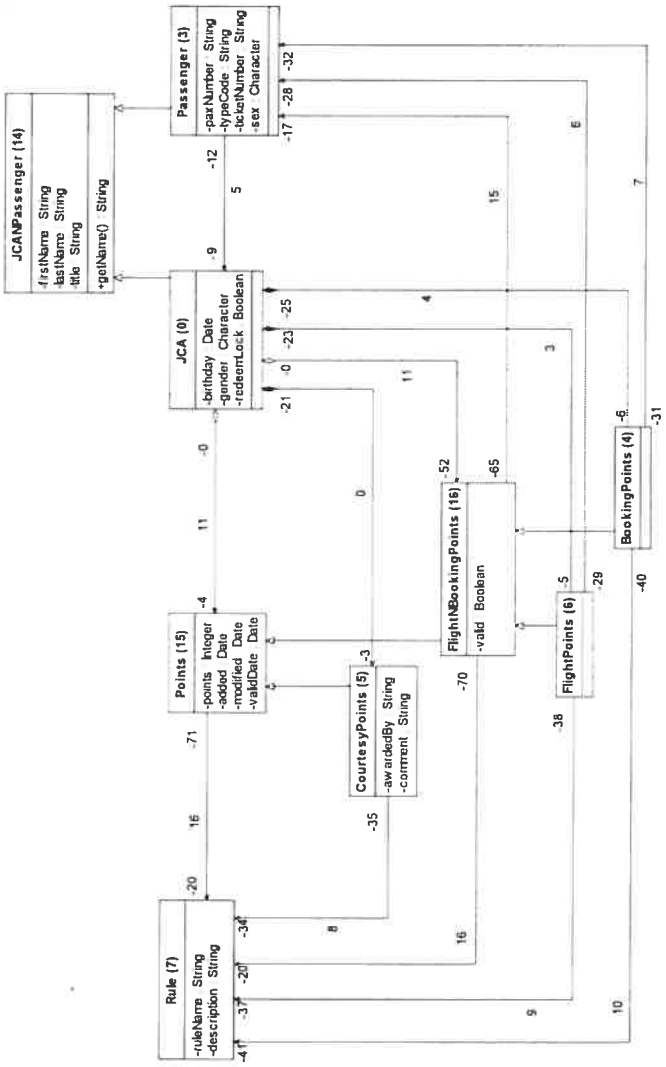


FIG. 8.7 – Correspondance entre les éléments du model UML final et les concepts des différents treillis .

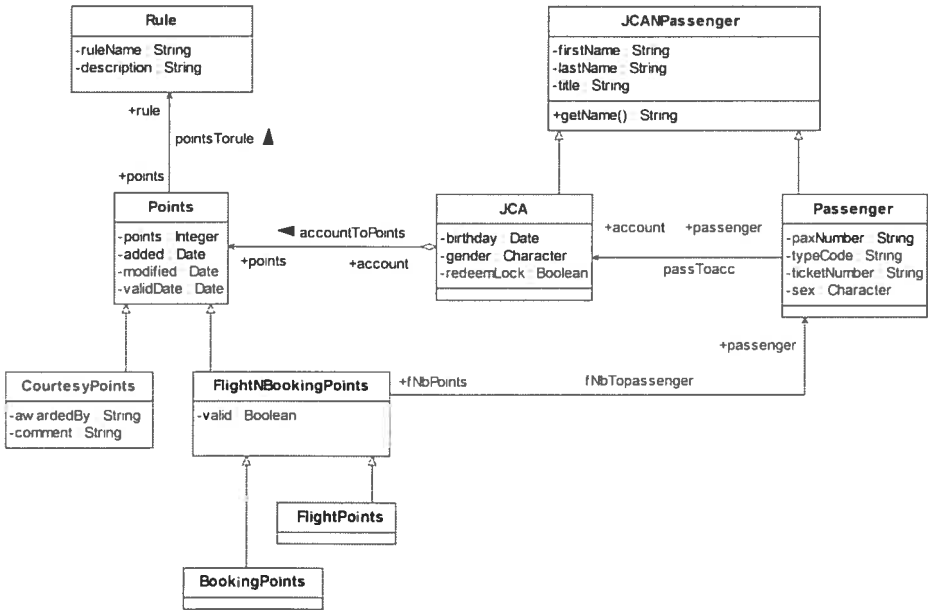


FIG. 8.8 – Modèle UML final qui correspond au modèle de la Figure 8.2. Les abstractions de rôles ont été supprimées par mesure de lisibilité.

Chapitre 9

Conclusion

L'AFC est un paradigme de découverte de connaissances basé sur la théorie des treillis dont la construction centrale est la correspondance de Galois entre un ensemble d'individus et un ensemble d'attributs. Les méthodes de l'AFC permettent d'extraire tous les groupes d'objets partageant des attributs (concepts) et de les ordonner dans un treillis. En génie logiciel, par exemple, l'AFC a permis de raisonner sur la structuration modulaire des programmes, d'en extraire des unités compréhensibles, d'analyser et de reconstruire des classes, des patrons de conception et des modèles de manière générale. Toutefois, le cadre fondamental de l'AFC ne permet pas de traiter les liens inter-individus tels que ceux qui existent entre les différents éléments d'un modèle UML. Ces liens constituent pourtant une source d'information très utile, notamment lors de la restructuration de tels modèles car la bonne factorisation des divers éléments du méta-modèle en dépend. L'introduction du traitement des liens inter-individus dans le paradigme fondamental de l'AFC ainsi que son utilisation effective dans la restructuration des modèles structuraux UML constituent les principales préoccupations de cette thèse.

9.1 Contributions

Nous avons proposé l'ARC comme un cadre général de traitement des informations relationnelles en AFC. L'ARC offre la possibilité d'avoir une famille de contextes (FCR), un par type d'individus, et des relations exprimant les liens inter-individus. Ces liens sont pris en compte lors de l'extraction des concepts en s'appuyant sur le mécanisme de scaling relationnel. De cette façon, les concepts obtenus reflètent non seulement le partage des attributs individuels des individus mais aussi des liens vers d'autres individus. Le processus d'extraction de concepts proposé conduit à la définition d'abstractions relationnelles qui décrivent de manière générique des liens inter-individus. Nous avons montré que ce processus converge et nous avons donné une expression analytique de la solution obtenue. Nous avons également donné les méthodes de base mettant en œuvre l'ARC, tels que l'algorithme de construction des treillis relationnels et l'algorithme de scaling relationnel.

La méthode de l'ARC a été implémentée au sein de la plateforme GALICIA et appliquée avec succès à la réorganisation des modèles structuraux UML. Pour ce faire, nous avons définit une transformation canonique d'un modèle UML en une FCR à partir d'une spécification XMI du modèle UML. Afin de résoudre les conflits de noms entre les éléments du modèles UML lors de la phase de transformation en une FCR, plusieurs solutions ont été mises en place dans l'outil RCFMODELER en s'appuyant sur des techniques de recherche d'information et sur l'utilisation de ressources linguistiques. Nous avons aussi proposé des algorithmes automatisant la phase post-ARC qui permet de produire un modèle UML à partir d'une famille de treillis.

9.2 ARC : portée et limites

Le cadre de l'ARC proposé s'appui sur l'AFC et est capable de traiter n'importe quelles données pouvant être structurées en un ensemble d'entités ayant des descriptions locales et des relations inter-entités à l'image du modèle E-R en modélisation

de données. La méthodologie de fouille de données relationnelles à l'aide de l'ARC consiste, d'abord, à coder les données du domaine d'application dans le format d'entrée de l'ARC, à savoir une FCR. La méthode de l'ARC est ensuite utilisée pour dériver une famille de treillis qui représentent l'ensemble des connaissances acquises à partir des données de départ. Dans une dernière étape, ces treillis sont analysés et les résultats de cette analyse sont interprétés dans le langage du domaine d'application.

L'étape de transformation initiale permettant de préparer les données à traiter peut occasionner une perte d'information -qui pourraient encore être pertinente pour la découverte de connaissances- dans le cas où le langage source a une sémantique plus riche que celle du langage de destination, en l'occurrence celui de l'ARC. De plus, l'analyse relationnelle s'appuie sur les treillis de concepts dont l'exploitation est une opération qui pourrait être fastidieuse car les données de taille industrielles conduisent, d'une part, à la dérivation de structures de grandes tailles d'où les difficultés de navigation, et d'autre part, à un grand nombre d'abstractions fortuites. Finalement, l'ARC pose le problème de la traduction des connaissances acquises au niveau des treillis de concepts en connaissances du domaine car il ne suffit pas de faire le rétro-codage des informations relationnelles au sein des concepts des différents treillis produits, mais aussi d'attribuer une sémantique de domaine aux concepts pertinents car du point de vue de l'AFC ces concepts ont été obtenus par une factorisation syntaxique de descriptions.

9.3 Perspectives

Nous pensons que ce travail doit être poursuivie selon deux volets, l'un théorique et l'autre applicatif. Sur le plan de la recherche fondamentale, divers problèmes restent posés par l'ARC. Tout d'abord, la formulation analytique du processus de construction de treillis gagnerait beaucoup si d'autres propriétés désirables sont exprimées et démontrées, telles que le nombre d'itérations avant convergence. L'accélération de la convergence du processus par la mise en place d'algorithmes

plus efficaces doivent aussi être étudiés. L'enjeu est tout autant théorique que pratique car cela nous permet de traiter de grandes quantités de données. La nécessité de faire intervenir un utilisateur dans le processus demande quant à elle la définition d'une manière d'introduire des points d'interruption dans les calculs.

De point de vue recherche applicative et vu l'importance accrue en génie logiciel de l'ingénierie des modèles[1], il faut approfondir et instrumentaliser les transformations endogènes (sans changement du langage de modélisation) qui visent à améliorer la qualité des modèles 'métier', en particulier en cherchant de meilleures abstractions. Les expériences de validation conduites sur les modèles UML ont mis en évidences différentes possibilités d'amélioration de la méthode, par exemple, lors de la transformation des données de départ dans le formalisme ARC. Les informations portées par les modèles ont des degrés d'importance sémantique variables et il s'agit donc de caractériser ces degrés et de les prendre en compte dans le processus. De plus, durant cette transformation une difficulté est introduite par l'utilisation de la langue naturelle dans le nommage des entités du modèle ce qui demande un recours à des outils linguistiques tels que des dictionnaires et des analyseurs pour la résolution des conflits de noms afin d'améliorer la pertinence des résultats. Bien qu'une étude initiale a été réalisée et a démontré l'efficacité des techniques choisies, nous pensons qu'un effort supplémentaire doit être investi dans ce sens afin d'améliorer la précision des résultats. De plus, la validation préliminaire sur la restructuration de modèles UML industriels a mis en lumière plusieurs limitations des outils actuels de l'approche notamment la capacité d'aide à la navigation et l'analyse des treillis de grande taille. En effet, dans le contexte de la restructuration des modèles UML par exemple, il y a un besoin de guider l'utilisateur lors de l'exploration des résultats de l'analyse car si l'AFC permet d'identifier toutes les abstractions potentiellement intéressantes, souvent seul un sous-ensemble d'entre elles s'avère vraiment pertinent.

Un autre aspect de l'ARC mérite d'être abordé. Il s'agit du rapprochement avec les logiques de descriptions. En effet, l'étude préliminaire effectuée dans le

[1]L'initiative MDA (Model Driven Architecture) de l'OMG (Object Management Group).

cadre de cette thèse a révélé l'intérêt de la méthode de l'ARC dans l'extraction des connaissances en logiques de descriptions. Ainsi, la correspondance entre le formalisme de l'ARC et celui des logiques de descriptions doit être approfondie afin que les résultats de l'analyse relationnelle puissent être directement insérés dans une base de connaissances à l'aide d'un langage de descriptions.

Bibliographie

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data, Washington (DC), USA*, pages 207–216, May 1993.

[2] R. S. Arnold. Software Reengineering. *IEEE Computer Society Press*, 1992.

[3] F. Baader and W. Nutt. Basic description logics. In Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook : Theory, Implementation, and Applications*, pages 43–95. Cambridge University Press, 2003.

[4] F. Baader and B. Sertkaya. Applying formal concept analysis to description logics. In Peter Eklund, editor, *Concept Lattices : Second International Conference on Formal Concept Analysis, ICFCA 2004, Sydney, Australia, February 23-26*, volume 2961 of *Lecture Notes in Computer Science*, pages 261–286. Springer Verlag, 2004.

[5] M. Barbut and B. Monjardet. *Ordre et Classification : Algèbre et combinatoire*. Hachette, 1970.

[6] C. Batini, M. Lenzerini, and S. B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Comput. Surv.*, 18(4) :323–364, 1986.

[7] K. H. Bennett. Legacy systems : Coping with success. *IEEE Software*, 12(1) :19–23, 1995.

[8] G. Birkhoff. *Lattice Theory*, volume XXV of *AMS Colloquium Publications*. AMS, 1940.

[9] G. Booch and al. *Object Oriented Analysis and Design*. Addison-Wesley, 1994.

[10] J. P. Bordat. Calcul pratique du treillis de Galois d'une correspondance. *Mathématique et Sciences Humaines*, (96) :31–47, 1986.

[11] Alexander Budanitsky and Graeme Hirst. Semantic distance in wordnet : An experimental, application-oriented evaluation of five measures. In *In Workshop on WordNet and other Lexical Resources, Second meeting of the Nord American Chapter of the Association for Computational Linguistics*, Pittsburgh, PA, USA, 2001.

[12] C. Carpineto and G. Romano. A Lattice Conceptual Clustering System and Its Application to Browsing Retrieval. *Machine Learning*, 24(2) :95–122, 1996.

[13] L. Chaudron and N. Maille. First order logic formal concept analysis : from logic programming to theory. *Computer and Information Science*, 13(3), 1998.

[14] P.P.-S. Chen. *Entity-Relationship Model : Towards a Unified View of Data*. North-Holland, Amsterdam, 1979.

[15] Chikofsky E.J. Cross and J.H. II. Software Reengineering. *Software, IEEE*, 7 :13–17, 1990.

[16] M. Dao, M. Huchard, M. Rouane Hacene, C. Roume, and P. Valtchev. Improving Generalization Level in UML Models : Iterative Cross Generalization in Practice. In H. Delugach K. E. Wolff, H. Pfeiffer, editor, *Proceedings of the 12th Intl. Conference on Conceptual Structures (ICCS'04)*, volume 3127 of *Lecture Notes in Computer Science*, pages 346–360. Springer Verlag, 2004.

[17] B. A. Davey and H. A. Priestley. *Introduction to lattices and order*. Cambridge University Press, 1992.

[18] H. Dicky, C. Dony, M. Huchard, and T. Libourel. ARES, un algorithme d'Ajout avec REStructuration dans les hiérarchies de classes. In F. Rechenmann, editor, *Actes du Colloque Langages et Modèles à Objets (LMO'94), Grenoble, France*, pages 125–136. INRIA Rhônes-Alpes – IMAG-LIFIA, Grenoble, 1994.

[19] 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.

[20] E. Diday and R. Emillion. Treillis de Galois maximaux et capacités de Choquet. *C.R. Acad. Sci. Paris*, 325(1) :261–266, 1997.

[21] V. Duquenne. Lattice structures in data analysis. 217 :407–436, 1999.

[22] S. Džeroski and N. Lavrač. *Relational Data Mining.* Springer, 2001.

[23] Jérôme Euzenat and Petko Valtchev. Similarity-based ontology alignment in owl-lite. In *ECAI '04 : Proceedings of the 16th European Conference on Artificial Intelligence. Valencia (SP).* IOS Press, Amsterdam, 2004.

[24] Christiane Fellbaum. *Wordnet : An Electronic Lexical Database.* MIT Press, 1998.

[25] S. Ferre and O. Ridoux. A logical generalization of formal concept analysis. In *Proceedings of the International Conference on Conceptual Structures, ICCS'2000,* 2000.

[26] M. Fowler. *Refactoring : Improving the Design of Existing Code.* Addison Wesley, 1999.

[27] J. Rumbaugh G. Booch and I. Jacobson. *The Unified Modeling Language User Guide.* Addison Wesley, 1999.

[28] B. Ganter. Two basic algorithms in concept analysis. preprint 831, Technische Hochschule, Darmstadt, 1984.

[29] B. Ganter, J. Stahl, and R. Wille. Measurement and many-valued contexts. *W. Gaul, M. Schader : Classification as a tool of research,* pages 169–176, 1986.

[30] B. Ganter and R. Wille. *Formal Concept Analysis. Mathematical Foundations.* Springer-Verlag, 1999.

[31] R. Girard. *Classification Conceptuelle sur des Données Arborescentes et Imprécises.* Thèse de doctorat, Université de la Réunion, 1997.

[32] R. Godin and H. Mili. Building and maintaining analysis-level class hierarchies using Galois lattices. In *Proceedings of OOPSLA'93, Washington (DC), USA,* special issue of ACM SIGPLAN Notices, 28(10), pages 394–410, 1993.

[33] R. Godin, H. Mili, G. Mineau, and R. Missaoui. Conceptual Clustering Methods Based on Galois Lattices and Applications. 9(2) :105–137, 1995.

[34] 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.

[35] R. Godin, H. Mili, G. W. Mineau, R. Missaoui, A. Arfi, and T. Chau. Design of class hierarchies based on concept lattices. *Theory and Application of Object Systems 4(2)*, pages 117–134, 1998.

[36] R. Godin, R. Missaoui, and H. Alaoui. Incremental Concept Formation Algorithms Based on Galois (Concept) Lattices. *Computational Intelligence*, 11(2) :246–267, 1995.

[37] V. Haarslev and R. Moller. Racer system description, 2001.

[38] I. Horrocks. The fact system, 1998.

[39] M. Huchard, H. Dicky, and H. Leblanc. Galois lattice as a framework to specify algorithms building class hierarchies. *Theoretical Informatics and Applications*, 34 :521–548, January 2000.

[40] M. Huchard, M. Rouane Hacene, C. Roume, and P. Valtchev. Relational Concept Discovery in Structured Datasets. *submitted to Discrete Applied Mathematics*, 2004.

[41] M. Huchard, C. Roume, and P. Valtchev. When concepts point at other concepts : the case of uml diagram reconstruction. In *In Proceedings of the Workshop FCAKDD*, pages 32–43, 2002.

[42] I. Jacobson J. Rumbaugh and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley Object Technology Series, 1999.

[43] I. Jacobson, M. Christerson, P. Jonsson, and G. Övergaard. *Object-Oriented Software Engineering : A Use Case Driven Approach*. Addison-Wesley, Reading (MA), USA, 1992.

[44] Mohamed Touzani Petko Valtchev Jérôme Euzénat, David Loup. Ontology alignment with ola. In *Proceedings of the 3rd EON Workshop, 3rd International Semantic Web Conference (ISWC), Hiroshima (JP)*, 2004.

[45] M. Munro K. H. Bennett, B. J. Cornelius and D. J. Robson. *Software Maintenance*. Butterworth Heinemann, 1991.

[46] R. Kent. Rough concept analysis : A synthesis of rough sets and formal concept analysis. *Funadamenta Informaticae*, 27 :169–181, 1996.

[47] S. Kuznetsov and S. Ob'edkov. Algorithms for the Construction of the Set of All Concept and Their Line Diagram. preprint MATH-AL-05-2000, Technische Universität, Dresden, June 2000.

[48] H. Leblanc. *Sous-hiérarchies de Galois : un modèle pour la construction et l'evolution des hiérarchies d'objets (Galois sub-hierarchies : a model for construction and evolution of object hierarchies).* PhD thesis, Université Montpellier 2, 2000.

[49] M. M. Lehman and L. A. Belady. *Program evolution.* Academic Press, New York, 1985.

[50] K. J. Lieberherr, P. Bergstein, and I. Silva-Lepe. From Objects to Classes : Algorithms for Optimal Object-Oriented Design 6(4). *Software Engineering*, 6 :205–228, 1991.

[51] C. Lindig and G. Snelting. Assessing modular structure of legacy code based on mathematical concept analysis. In ACM Press, editor, *19th International Conference on Software Engineering Software Engineering (ICSE-19)*, 1997.

[52] B. Meyer. *Object-Oriented Software Construction.* Prentice Hall, second edition, 1997.

[53] G. W. Mineau, G. Stumme, and R. Wille. Conceptual structures represented by conceptual graphs and formal concept analysis. In W. M. Tepfenhart and W. Cyre, editors, *Conceptual structures : Standards and Practices*, volume 1640, pages 423–441, 1999.

[54] B. Nebel. Terminological cycles : Semantics and computational properties. In J. F. Sowa, editor, *Principles of Semantic Networks : Explorations in the Representation of Knowledge*, pages 331–361. Morgan Kaufmann Publishers, San Mateo (CA), USA, 1991.

[55] K. Nehme, P. Valtchev, M. H. Rouane, and R. Godin. On computing the minimal generator family for concept lattices and icebergs. In B. Ganter and R. Godin, editors, *Proceedings of the 3rd Intl. Conference on FCA, Lens (FR), February 14-18*, LNCS 3403, pages 192–207. Springer Verlag, 2005.

[56] K. Nehme, P. Valtchev, M.H. Rouane, and R. Godin. On computing the minimal generator family for concept lattices and icebergs. In *International Conference on Formal Concept Analysis (ICFCA'2005)*, Lens,France, 2005.

[57] Kamal Nehmé. Maîtrise en informatique. Mise à jour de la famille des générateurs minimaux des treillis de concepts et des icebergs. Décembre 2004.

[58] L. Nourine and O. Raynaud. A Fast Algorithm for Building Lattices. *Information Processing Letters*, 71 :199–204, 1999.

[59] P. F. Patel-Schneider and B. Swartout. Description logic specification from the krss. http ://www.bell-labs.com/user/pfps/papers/krss-spec.ps (2003).

[60] Tom Pender. *UML Bible*. John Wiley & Sons, Inc., New York, NY, USA, 2003.

[61] Gregory Piateski and William Frawley. *Knowledge Discovery in Databases*. MIT Press, Cambridge, MA, USA, 1991.

[62] S. Prediger. Logical scaling in formal concept analysis. In *International Conference on Conceptual Structures*, pages 332–341, 1997.

[63] S. Prediger and G. Stumme. Theory-driven logical scaling. In *International Workshop on Description Logics, Sweden*, volume 22, 1999.

[64] Susanne Prediger and Rudolf Wille. The lattice of concept graphs of a relationally scaled context. In *Proceedings of the 7th International Conference on Conceptual Structures*, pages 401–414. Springer-Verlag, 1999.

[65] U. Priss. *Relational Concept Analysis : Semantic Structures in Dictionaries and Lexical Databases*. PhD thesis, Aachen University, 1996.

[66] T. Quatrani. Introduction to the Unified Modeling Language. *Rational Developer Network, Rational Software*, 2001.

[67] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal : Very Large Data Bases*, 10(4) :334–350, 2001.

[68] M. H. Rouane, K. Nehme, P. Valtchev, and R. Godin. On-line maintenance of iceberg concept lattices. In *Contributions to the 12th ICCS*, page 14 p., Huntsville (AL), 2004. Shaker Verlag.

[69] M. Hacene Rouane. Magalice : An incremental algorithm for iceberg lattice maintenance. In *Workshop, Laval, France*, 2003.

[70] Cyril Roume. *Analyse et restructuration de hiérarchies de classes*. PhD thesis, Université de Montpellier II, 2004.

[71] S. Rudolph. Exploring relational structures via fle. In K. E. Wolff, H. D. Pfeiffer, and H. S. Delugach, editors, *Conceptual Structures at Work : 12th International Conference on Conceptual Structures, ICCS 2004, Huntsville, AL, USA, July 19-23,*

2004, volume 3127 of *Lecture Notes in Computer Science*, pages 196–212. Springer Verlag, 2004.

[72] J. Rumbaugh and al. *Object Oriented Modeling and Design*. Prentice Hall, 1991.

[73] H. A. Sahraoui, H. Lounis, W. Melo, and M. MILI. A Concept Formation Based Approach to Object Identification in Procedural Code. *Automated Software Engineering Journal*, 6 :387–410, 1999.

[74] M. Siff and T. Reps. Identifying modules via concept analysis. In IEEE Computer Society, editor, *In International Conference on Software Maintenance (ICSM97)*, 1997.

[75] G. Snelting. Reengineering of configurations based on mathematical concept analysis. *ACM Transactions on Software Engineering and Methodology 5(2)*, pages 146–189.

[76] G. Snelting. a new framework for program understanding. In SIGPLAN Notices 33(7), editor, *Proceedings of the ACM SIGPLAN/SIGSOFT Workshop on Program Analysis for Software Tools and Engineering (PASTE'98)*, pages 349–359, 1998.

[77] G. Snelting. Software reengineering based on concept lattices. In IEEE Computer Society, editor, *Proceedings of the 4th European Conference on Software Maintenance and Reengineering (CSMR'00)*, 2000.

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

[79] Mirko Streckenbach and Gregor Snelting. Refactoring class hierarchies with kaba. In *OOPSLA '04 : Proceedings of the 19th annual ACM SIGPLAN Conference on Object-oriented programming, systems, languages, and applications*, pages 315–330, New York, NY, USA, 2004. ACM Press.

[80] G. Stumme, R. Taouil, Y. Bastide, N. Pasquier, and L. Lakhal. Computing Iceberg Concept Lattices with Titanic. *Data and Knowledge Engineering*, 42(2) :189–222, 2002.

[81] T. Tilley, R. Cole, P. Becker, and P. Eklund. A survey of formal concept analysis support for software engineering activities. In *Proceedings of the First International*

Conference on Formal Concept Analysis, Darmstadt, Germany. Springer Verlag, February 2003.

[82] Paolo Tonella and Giulio Antoniol. Object oriented design pattern inference. In *ICSM '99 : Proceedings of the IEEE International Conference on Software Maintenance*, page 230, Washington, DC, USA, 1999. IEEE Computer Society.

[83] P. Valtchev. An algorithm for minimal insertion in a type lattice. *Computational Intelligence*, 15(1) :63–78, 1999.

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

[85] P. Valtchev, M. Rouane Hacene, D. Grosser, C. Roume, R. Godin, and R. Missaoui. Galicia home page : http ://www.iro.umontreal.ca/~galicia/.

[86] P. Valtchev, M. Rouane Hacene, and R. Missaoui. A generic scheme for the design of efficient on-line algorithms for lattices. In A. de Moor, W. Lex, and B. Ganter, editors, *Proceedings of the 11th Intl. Conference on Conceptual Structures (ICCS'03)*, volume 2746 of *Lecture Notes in Computer Science*, pages 282–295, Berlin (DE), 2003. Springer-Verlag.

[87] P. Valtchev and R. Missaoui. Building concept (Galois) lattices from parts : generalizing the incremental methods. In H. Delugach and G. Stumme, editors, *Proceedings of the ICCS'01*, volume 2120 of *Lecture Notes in Computer Science*, pages 290–303, 2001.

[88] P. Valtchev, R. Missaoui, and R. Godin. A Framework for Incremental Generation of Closed Itemsets. *accepted in Discrete Applied Mathematics*.

[89] P. Valtchev, R. Missaoui, and R. Godin. A framework for incremental generation of frequent closed itemsets. In *Proceedings of the 1st International Workshop on Discrete Mathematics and Data Mining*, Washington (DC), USA, April 2002.

[90] P. Valtchev, R. Missaoui, R. Godin, and M. Meridji. Generating Frequent Itemsets Incrementally : Two Novel Approaches Based On Galois Lattice Theory. *Journal of Experimental & Theoretical Artificial Intelligence*, 14(2-3) :115–142, 2002.

[91] P. Valtchev, R. Missaoui, and P. Lebrun. A partition-based approach towards building Galois (concept) lattices. *Discrete Mathematics*, 256(3) :801–829, 2002.

[92] Petko Valtchev. *Construction automatique de taxonomies pour l'aide à la représentation de connaissances par objets.* PhD thesis, Université Joseph Fourier, Grenoble I, November 1999.

[93] A. van Deursen and T. Kuipers. Identifying objects using cluster and concept analysis. In ACM Press, editor, *21st International Conference on Software Engineering, (ICSE-99)*, pages 246–255, 1999.

[94] F. Vogt and R. Wille. TOSCANA – a graphical tool for analyzing and exploring data. In R. Tamassia and I. G. Tollis, editors, *Graph Drawing*, volume 894, pages 226–233, 1994.

[95] R. Wille. Restructuring lattice theory : an approach based on hierarchies of concepts. *Ordered Sets*, pages 445–470, 1982.

[96] R. Wille. Concept Lattices and Conceptual Knowledge Systems. *Computers and Mathematics with Applications*, 23 :493–515, 1992.

[97] Xifeng Yan and Jiawei Han. Closegraph : mining closed frequent graph patterns. In *KDD '03 : Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 286–295, New York, NY, USA, 2003. ACM Press.

Annexe A

La plateforme Galicia

Vue Générale de Galicia 2.0

GALICIA permet de réaliser les opérations standards de manipulations et de visualisation de contextes et des treillis. Elle permet aussi de calculer les règles d'association d'un contexte donné et de les visualiser. D'autres opérations de support sont aussi offertes telles que la gestion d'un entrepôt de données et l'exportation des données et des résultats en différents formats.

Manipulation des contextes

La plateforme offre diverses opérations de manipulation de contextes, telles que l'import, l'export et l'édition interactive des données, le découpage en fragment, etc. Outre les contextes binaires classiques, des contextes multi-valués sont possibles lors de la conception, de même que des descriptions de données plus complexes (i.e. les FCR précitées). Un large éventail de formats de données propriétaires sont proposés dans la plateforme. La Figure 1 présente le style d'édition de données de GALICIA.

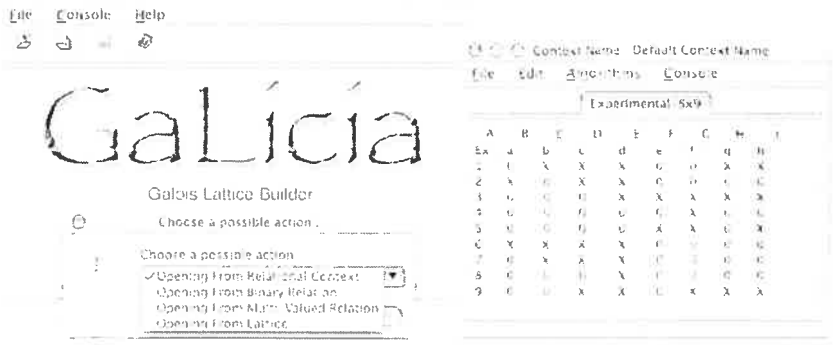


FIG. 1 – Édition de contextes dans GALICIA.

Construction et maintenance de treillis

La plateforme offre plusieurs algorithmes de construction de treillis. La plateforme fournit également un ensemble d'algorithmes de construction de structures dérivées de treillis tels que TITANIC [80], MAGALICE-A [56] et MAGALICE-O [68] pour les icebergs et CERES [18] et ARES [18] pour les SHG. Les méthodes proposées dans le cadre du paradigme de l'ARC pour la construction de treillis à partir d'une FCR sont aussi implémentées dans la plateforme. Les générateurs minimaux [2] des intensions de concepts peuvent être obtenus [57]. Enfin, GALICIA incorpore aussi un ensemble d'algorithmes de construction de règles d'association [88, 89].

Les méthodes de construction "diviser pour régner" ont été inclues dans la plateforme permettant ainsi la construction de treillis par assemblage de treillis partiels (apposition/subposition de contextes). La Figure 2 illustre un treillis construit par la plateforme en utilisant l'agorithme de Valtchev et *al* [87].

[2] Étant donnée une intension, un générateur minimal est un sous-ensemble minimal d'attributs formels dont la présence dans une description d'objet formel entraîne la présence de tous les attributs de cette intension.

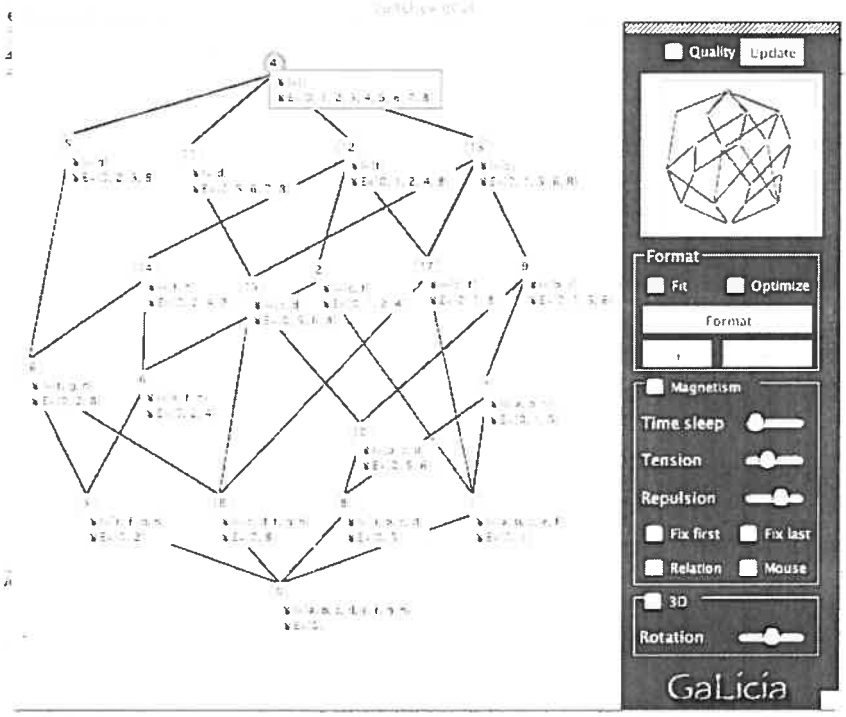


FIG. 2 – Construction de treillis de Galois.

Visualisation des treillis

La représentation graphique du treillis est un des moyens les plus pratiques pour communiquer les résultats de l'AFC. GALICIA offre deux méthodes de mise en forme du treillis, l'une entièrement automatique, l'autre interactive. La première méthode s'appuie sur l'approche des diagrammes par niveaux où chaque sommet est assigné à un niveau horizontal. Pour un niveau donné, les nœuds sont ordonnés de manière à minimiser le nombre de croisements des arêtes. La seconde méthode permet d'observer le treillis en trois dimensions et d'effectuer des rotations. La Figure 3 montre un treillis en 2 dimensions.

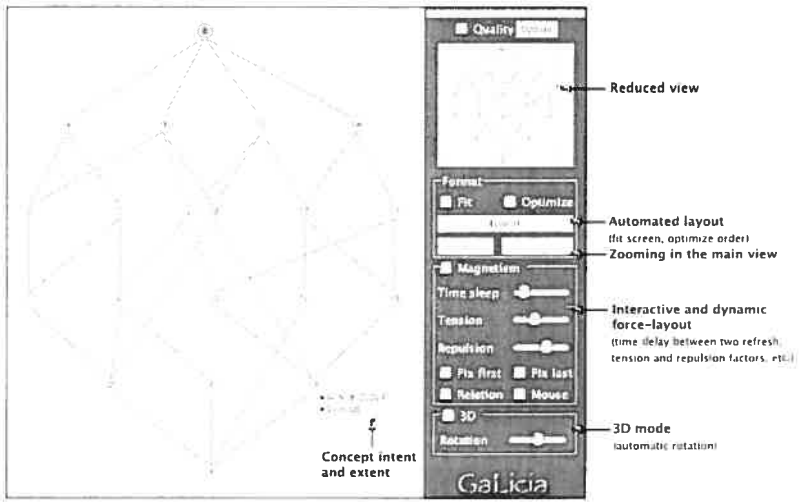


FIG. 3 – Visualisation des treillis en 2D.

Gestion de données

GALICIA permet de stocker les données d'entrées (contextes binaires, FCR, etc.) et les résultats (treillis, règles d'association, etc.) dans une base de données relationnelle MySQL. La Figure 4 montre les paramètres permettant d'établir une connexion à une base de données distante, à savoir, l'adresse du serveur de la base, le nom de la base, le nom de l'utilisateur et le mot de passe.

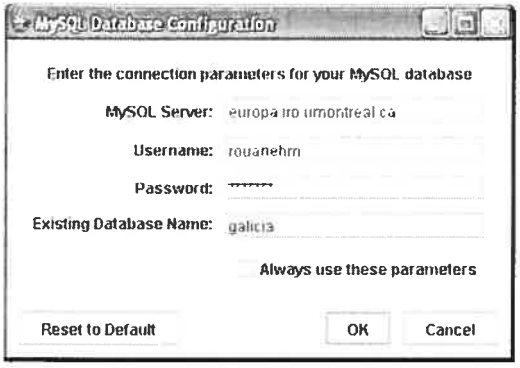


FIG. 4 – Connexion à une base de données MySQL avec GALICIA.

GALICIA offre aussi une interface de saisie de requêtes SQL. Les requêtes SQL sont un moyen efficace de recherche dans les treillis volumineux ou encore les grandes bases de règles d'association.

Exportation des résultats en une base LD

Les travaux sur l'ARC et les logiques de description ont conduit à l'élaboration d'une correspondance entre les concepts des deux disciplines(voir Chapitre 4, Section 4.8). GALICIA offre la possibilité d'exporter les treillis relationnels obtenus par la méthode MULTI-FCA en une base de connaissances en logique de descriptions. À l'heure actuelle, le format KRSS[3] [59] est utilisé pour l'expression des connaissances. Ce format est interprétable par la plupart des systèmes de LD tel que RACER[4] [37]. La Figure 5 illustre la base de connaissances LD en format KRSS générée par GALICIA à partir du treillis relationnel de la Figure 12.

Un fichier KRSS est composé de trois principales sections, à savoir, la section qui spécifie la signature de la base, la section qui donne les définitions des concepts de la base (concepts logiques) et la section qui donne les extensions des concepts et des rôles. La première section représente l'aspect terminologique de la base de connaissances et permet de déclarer les concepts atomiques, les rôles et les individus de la base. Dans

[3]Description-Logic Knowledge Representation System Specification.

[4]http ://www.racer-systems.com

l'exemple de la Figure 5, il y a un concept logique atomique pour chaque concept formel du treillis de la Figure 12. Les noms des concepts logiques sont générés à partir du nom du contexte d'origne et du numéro du concept dans le treillis associé. Comme la RCF dispose d'une seule relation, en l'occurence `spouse`, la base possède un seul rôle qui correspond à cette relation. Finalement, les individus de la base correspondent aux objets formels de l'unique contexte `Human`. La seconde section représente l'aspect factuel ou assertionnel. Les concepts définis dans cette section sont introduits par le mot clé `implies` qui exprime la relation de subsumption (inclusion générale) entre deux concepts logiques. Le constructeur `and` indique qu'un concept est construit à partir d'une conjonction de concepts, qui sont les ascendants directs du nouveau concept. Le constructeur `all` précise le co-domaine du rôle. Dans l'exemple de la Figure 5, le concept logique `Human-C0` désigne tous les individus dont l'âge est égal à 25 ans, qui représentent des instances du concept logique `Human-C6` et dont tous les conjoints sont des instances du concept logique `Human-C3`. La troisième section donne les extensions des différents concepts logiques et rôles en utilisant les constructeurs `instance` et `related`, respectivement. Par exemple, tous les individus `01...06` sont des instances du concept logique `Human` qui subsume tous les autres concepts logiques.

Conception de Galicia

GALICIA possède une architecture en trois couches qui a été définie au dessus de la librairie standard Java 1.4. Cette architecture (voir Figure 6) reflète le désir d'adaptabilité, d'extensibilité et de réutilisabilité qui motive son développement.

La couche la plus basse de la plateforme définit le noyau du système : un ensemble de types abstraits de données et d'implémentations concrètes de ces types, correspondant aux différentes structures manipulées par GALICIA (concepts, treillis, contextes, relations, règles d'association). La couche intermédiaire offre l'ensemble des outils de haut niveau assurant le cycle de vie d'un treillis. Le premier outil est celui qui permet l'édition des différents types de contextes et de familles de contextes (FCR). L'outil `Structure extractor` permet la construction et/ou maintenance des structures conceptuelles telles que les treillis, SHG et icebergs. L'outil `Lattice viewer` permet l'affichage et la navigation dans les structures créées. Le dernier outil, `I/O`, assure le flux d'entrée/sortie

```
(in-knowledge-base Human)
(signature :atomic-concepts( Human age=22 age=25 age=26 age=28 age=33 age=35 e.t.=1 e.t.=2 e.t.=4 e.t.=6 e.t.=8
Human-C2 Human-C6 Human-C4 Human-C5 Human-C9 Human-C0 Human-C7 Human-C1 Human-C3 Human-C8)
:roles( ( spouse )) :individuals( O1 O2 O3 O4 O5 O6))

(implies Human Human) (implies Human-C2 (all spouse Human-C2))
(implies Human-C6 ( and e.t.=4 ( and (all spouse Human-C9) Human-C2)))
(implies Human-C4 ( and age=22 ( and e.t.=1 ( and (all spouse Human-C5) Human-C2))))
(implies Human-C5 ( and age=26 ( and e.t.=8 ( and (all spouse Human-C4) Human-C2))))
(implies Human-C9 ( and (all spouse Human-C6) Human-C2))
(implies Human-C0 ( and age=25 ( and (all spouse Human-C3) Human-C6)))
(implies Human-C7 ( and age=33 ( and (all spouse Human-C8) Human-C6)))
(implies Human-C1 ( and Human-C0 ( and Human-C8 ( and Human-C5 ( and Human-C4 ( and Human-C3 Human-C7))))))
(implies Human-C3 ( and age=28 ( and e.t.=2 ( and (all spouse Human-C0) Human-C9))))
(implies Human-C8 ( and age=35 ( and e.t.=6 ( and (all spouse Human-C7) Human-C9))))

(instance O1 Human) (instance O2 Human) (instance O3 Human) (instance O4 Human)
(instance O5 Human) (instance O6 Human)

(related O1 O2 spouse) (related O2 O1 spouse) (related O3 O4 spouse) (related O4 O3 spouse)
(related O5 O6 spouse) (related O6 O5 spouse)

%(save-kb "C:/human.owl" :tbox RCF :abox RCF :syntax :owl :uri "http://www.w3.org/2002/07/owl" :ns0
%"http://www.w3.org/2002/07/owl#")
```

FIG. 5 – Base de connaissances LD en KRSS générée par GALICIA à partir du treillis relationnel final du contexte Humain 12.

des données et des résultats. Finalement, la couche supérieure définit le code spécifique à l'interface avec l'usager en respectant les principes du modèle MVC (Modèle-Vue-Contrôleur).

L'architecture de GALICIA autorise l'intégration de différents outils à travers un ensemble d'interfaces de programmation standards définies dans la couche basse de la plateforme, tandis que les outils de haut niveau sont intégrés dans un environnement graphique uniforme.

Galicia et l'ARC

Les données relationnelles sont représentées au sein de GALICIA par le biais d'une FCR. La méthode MULTI-FCA (voir Chapitre 5, Section 5.1.2) a été implémentée au sein de GALICIA permettant la construction d'un ensemble de treillis inter-reliés à partir de cette FCR.

L'activité d'analyse de données relationnelles nécessite souvent plusieurs exécution de l'algorithme MULTI-FCA avec différents paramètres. Pour ce faire, GALICIA offre la

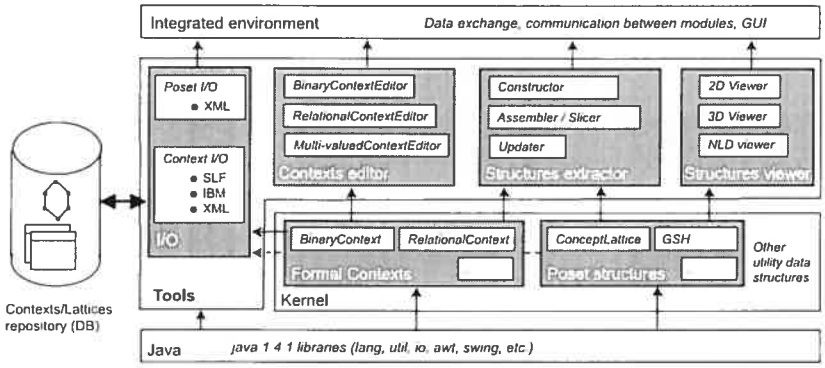


FIG. 6 – Conception architecturale de la plateforme GALICIA.

possibilité de choisir pour chaque exécution : les contextes et/ou relations qui forment la FCR, la structure conceptuelle que l'on désire obtenir (treillis complets ou SHG), le mode de scaling à utiliser (étroit ou large) et le style d'affichage des résultats (extensions/intensions completes ou réduites).

Composition de la FCR

GALICIA permet de charger ou de définir une famille de contextes avec des relations inter-contextes. Elle permet, par la suite de choisir les contextes et les relations qui vont être utilisés comme entrée pour la méthode MULTI-FCA. Ceci permet d'écarter des contextes et/ou relations qui sont moins utiles à l'analyse. La Figure 7 et la Figure 8 montrent le contexte humain et la relation conjoint. Les valeurs des attributs multi-valués sont données entre deux crochets. Par exemple, étant donné un contexte muni de l'attribut multi-valué 'couleurs-préférées' ; l'expression [rouge,vert] donne l'ensemble des valeurs de cet attribut pour un individu de ce contexte.

Les modes de scaling pour l'ARC

GALICIA 2.0 effectue un scaling nominal automatique des attributs formels multi-valués. La Figure 9 montre le scaling conceptuel nominal par GALICIA des deux attributs pluri-valués âge et expérience de travail du contexte humain.

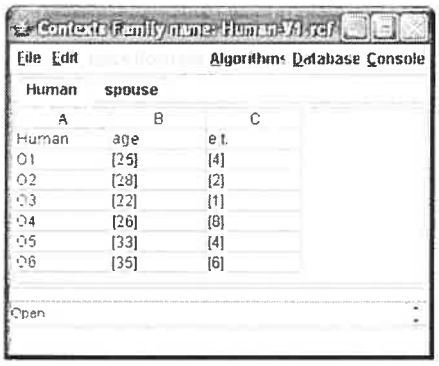


FIG. 7 – Le contexte humain dans GALICIA.

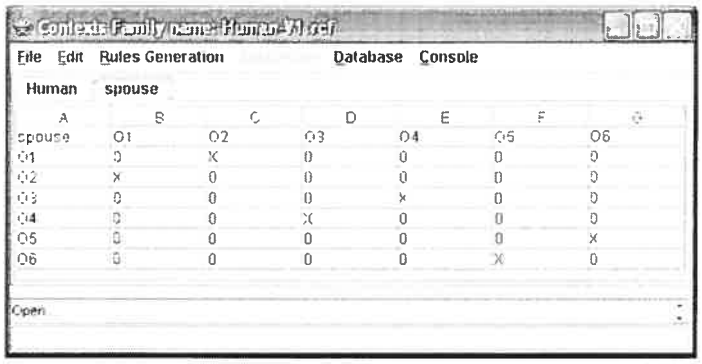


FIG. 8 – La relation conjoint entre les individus du contexte humain dans GALICIA.

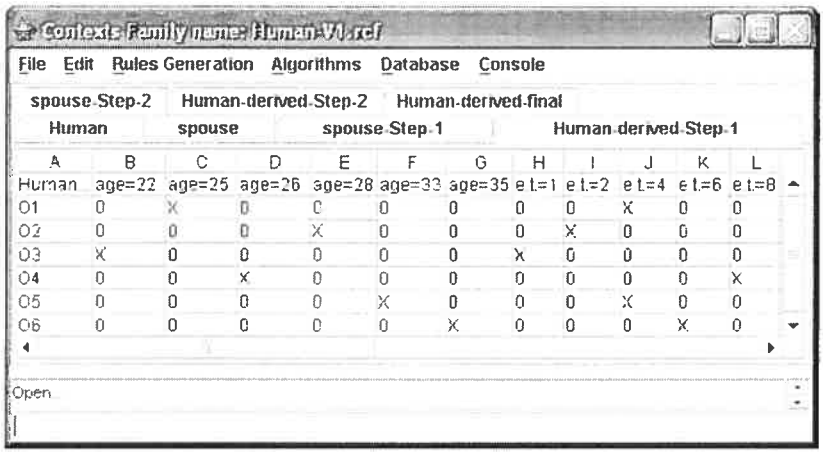


FIG. 9 – Scaling conceptuel nominal des attributs pluri-valués dans GALICIA.

Le scaling relationnel (voir Chapitre 4, Section 4.5.1) sur lequel se base la méthode MULTI-FCA est aussi implémenté. La Figure 10 montre le scaling de la relation conjoint durant la première étape de la méthode itérative MULTI-FCA. Comme illustré par cette figure, les attributs formels désignent des concepts du treillis initial du contexte de la Figure 7.

Le scaling relationnel sur lequel se base la méthode MULTI-FCA et qui consiste à utiliser des treillis comme une source d'abstraction pour la construction des échelles des relations inter-contextes est évidemment implémenté. La Figure 10 montre le scaling relationnel par GALICIA de la relation conjoint durant la première étape de la méthode itérative MULTI-FCA.

Paramétrage de l'algorithme Multi-FCA

GALICIA permet de définir tous les paramètres de l'algorithme MULTI-FCA allant du choix de la structure que l'on veut obtenir en fin du processus (pour le moment treillis ou SHG) jusqu'au style d'affichage des résultats (étiquetage complet ou réduit des intensions des concepts). Ci-dessous une brève description de quelques paramètres essentiels.

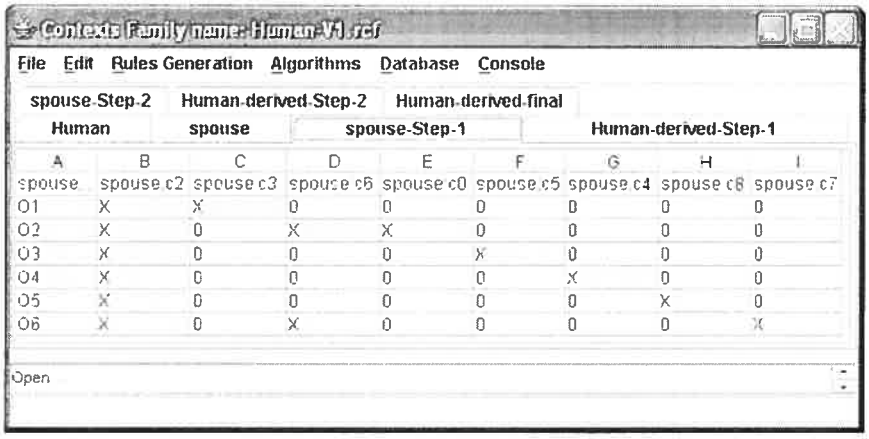


FIG. 10 – Scaling de la relation conjoint lors de la première itération de MULTI-FCA.

Choix de la structure cible

La Figure 11 illustre le choix de la structure conceptuelle que l'algorithme MULTI-FCA construit. Deux choix sont offerts, à savoir le treillis complet et la SHG. Toutefois, notre travail se concentre sur la construction du treillis. Il est à noter que la méthode MULTI-FCA utilise un algorithme incrémental par ajout d'attributs pour la maintenance de la structure conceptuelle cible. Dans le cas d'un treillis complet, l'algorithme incrémental a l'avantage de conserver l'identité des concepts. En effet, dès qu'un concept est créé, il reste dans le treillis jusqu'à la terminaison de la méthode bien que son intension peut changer au cours des itérations. L'algorithme de mise à jour incrémentale associe un nouveau identifiant à tout nouveau concept.

Cette propriété est une qualité très appréciée pour l'analyse car elle permet l'identification et le suivi des informations au cours des itérations. Cependant, les exécutions utilisant des treillis complets ont l'inconvénient de prendre plus de temps et d'occuper plus d'espace car les treillis peuvent être d'une grande taille ; chose qui pose des limites dans la lisibilité du treillis.

Les SHG par contre, ont l'avantage d'être des structures plus compactes d'où une grande économie en espace. Toutefois, l'opération de maintenance sous-jacente est non monotone dans la mesure où des concepts peuvent être détruits suite à l'ajout d'un attribut. Dans ces conditions, un même concept (extension) peut se voir affecter des

identifiants différents.

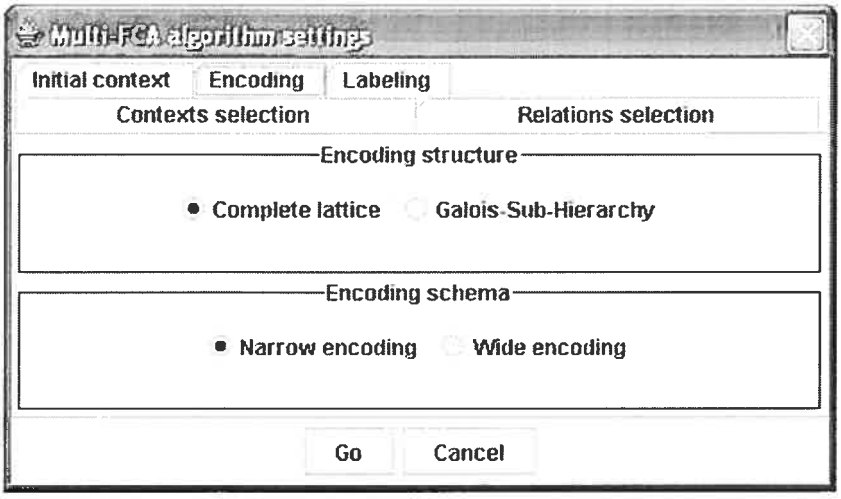


FIG. 11 – Paramétrage de la méthode MULTI-FCA dans la plateforme GALICIA.

Modes de scaling relationnel

Le mécanisme de scaling relationnel permet de ramener les descriptions relationnelles des individus en descriptions propositionnelles en s'appuyant sur des échelles inférées à partir des abstractions découvertes en cours d'exécution de la méthode MULTI-FCA. Autrement dit, au lieu d'utiliser des échelles pré-établies (obtenues à partir des données à l'étape de pré-traitement), MULTI-FCA a l'avantage d'utiliser des échelles dynamiques établies au fur et à mesure que de nouveaux résultats sont obtenus par le processus itératif. De plus, il y a deux manières de coder la relation entre un individu donné et une abstraction décrivant un groupe d'individus. En mode de codage étroit (narrow), la relation entre l'individu et l'abstraction existe si tous les individus auxquels l'individu en question est lié sont décrit par cette abstraction. En mode de codage large (wide), pour que cette relation existe, il suffit que certains individus auxquels l'individu en question est lié sont couverts par l'abstraction (pour plus de détails, voir Chapitre 4, Section 4.5.1).

À cet effet, GALICIA permet de préciser le mode de codage des liens inter-individus lors du scaling des relations par la méthode MULTI-FCA. La Figure 11 montre comment paramétrer MULTI-FCA afin qu'elle utilise un style de codage étroit.

Affichage des résultats

En plus des treillis finaux des différents contextes de la FCR, GALICIA permet d'avoir les contextes des échelles relationnelles et les treillis intermédiaires au bout de chaque itération. La Figure 12 illustre le treillis relationnel final obtenu par GALICIA à partir de la FCR composée du contexte humain (Figure 7) et de la relation conjoint (Figure 8). On remarque que les individus du contexte Humain ne partage aucun attribut initial à cause du scaling nominal des attributs initaux. Toutefois, grâce à la prise en compte de la relation conjoint dans la construction du treillis, tous les individus (concept-objets) indiquent leur conjoint. De plus, il y deux concepts, l'un est $c_{\#6}$ qui représente les traits commun entre les individus et l'autre est $c_{\#9}$ qui indique la description la plus générale des conjoints de ses membres.

Dans GALICIA, il est possible de choisir, au niveau de l'étape de paramétrage de l'algorithme, entre un étiquetage complet et un étiquetage réduit de la partie locale (attributs initiaux Int_l) et/ou la partie relationnelle (attributs obtenus par le scaling relationnel Int_r) des intensions des concepts.

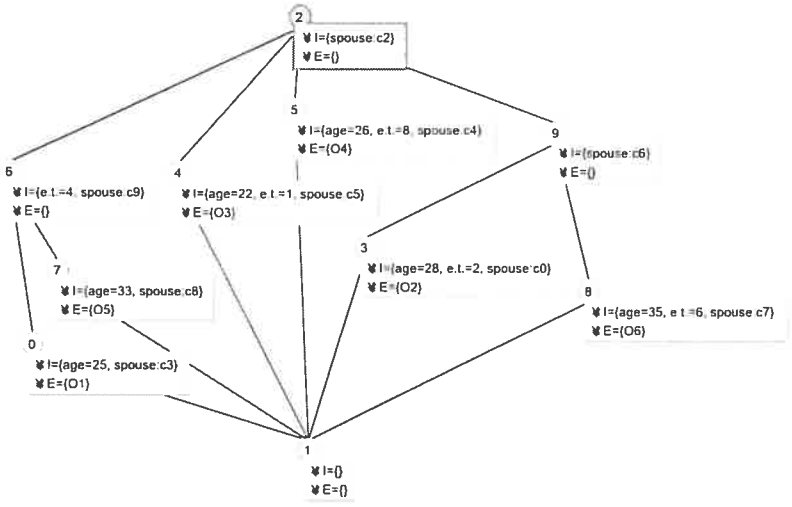


FIG. 12 – Le treillis relationnel final construit par GALICIA à partir du contexte humain (Figure 7) et de la relation conjoint (Figure 8).

Annexe B

Signification des caractéristiques des éléments UML

Caractéristique	Valeurs possibles	Signification
`name`	chaîne de caractères.	Un nom de classe.
`visibility`	public. private. protected. package.	`Public` : la classe est visible partout. `Private` : seulement pour les classes internes (inner) et n'est visible que dans les méthodes définies au sein de la classe de niveau supérieur (outer), `Protected` : visible dans les méthodes définies au sein de la classe englobante et de ses sous-classes. `Package` : visibilité limitée au package auquel la classe appartient.
`isAbstract`	true. false	Une classe est abstraite si sa déclaration est incomplète (par exemple, comporte des méthodes abstraites).
`isActive`	true. false	Précise si une classe comporte un processus d'exécution propre. Les objets de cette classe possèdent leur propre flot de contrôle et sont donc capables d'initier des activités.
`isDataType`	true, false	La classes représente un type de données ou pas.
`isClass`	true. false	L'élément représente une classe ou pas.

TAB. 1 Signification des caractéristiques de la classe en UML.

Caractéristique	Valeurs possibles	Signification
`name`	chaîne de caractères.	Un nom d'attribut. Il doit être unique parmi les attributs hérités et les rôles d'associations sortantes.
`visibility`	public, private, protected, package.	Précise si l'attribut peut être utilisé par les autres classes. La visibilité des classes imbriquées est combiné et la plus restrictive est gardée. Public : toute classe externe peut utilisé l'attribut. Protected : tout descendant de la classe peut utilisé l'attribut. Private : seulement la classe a accès à l'attribut. Package : toute classe déclarée dans le même package (ou sous-package de n'importe quel niveau) peut utilisé l'attribut.
`initialValue`	valeur	Une valeur du domaine de déclaration de l'attribut.
`multiplicity`	[1..1], [1..N], etc.	le nombre possible de valeur de données que l'attribut peut avoir pour une instance données. La cardinalité de l'ensemble des valeurs constitute une partie implicite de l'attribut. Par exemple, dans le cas ou l'attribut est scalaire, c'est à dire, renferme une seule valeur, la multiplicité est de type 1..1.
`changeability`	changeable, frozen, addOnly	Précise si la valeur de l'attribut peut être modifiée après l'initialisation de l'objet une classe comporte un processus. Changeable : aucune restriction sur la modification. Frozen : aucune valeur ne peut être ajoutée ou retirée après l'initialisation de l'objet. AddOnly : des valeurs peuvent être ajoutées à tout moment et aucune valeur ne peut être retirée après l'initialisation.

TAB. 2 – Signification des caractéristiques d'un attribut de classe en UML.

Caractéristique	Valeurs possibles	Signification
`ownerScope`	instance, classifier	Spécifie si chaque instance de la classe possède sa propre occurrence de l'attribut ou bien toutes les instances de la classe partage une seule occurrence de cet attribut (attribut `static` en Java). Instance : chaque instance de la classe possède sa propre valeur de l'attribut. Classifier : il y a une seule valeur pour tous les objets instanciés à partir de cette classe.
`targetScope`	instance, classifier	Spécifie si la cible est une instance ordinaire ou une classe. Instance : chaque valeur contient une référence vers une instance de la classe cible. Classifier : chaque valeur contient une référence vers la classe cible.

TAB. 3 – La suite de la signification des caractéristiques d'un attribut de classe en UML.

Caractéristique	Valeurs possibles	Signification
`name`	chaine de caractères.	La signature complète de l'opération.Il doit être unique au sein de la classe.
`visibility`	public, private, protected, package.	Voir la Table 2
`isAbstract`	true, false	Méthode déclarée mais pas encore définie.
`concurrency`	true, false	Précise si une classe comporte un processus d'exécution propre. Les objets de cette classe possèdent leur propre flot de contrôle et sont donc capables d'initier des activités.
`isQuery`	true, false	Précise si l'exécution de l'opération modifie l'état du système. True : indique que l'état demeure inchangée. False : indique que des effets de bord peuvent surgir.
`ownerScope`	instance, classifier	Voir la Table 3

TAB. 4 – Signification des caractéristiques d'une opération de classe en UML.

Caractéristique	Valeurs possibles	Signification
`name`	chaîne de caractères.	Un nom de rôle. Il représente un pseudo-attribut de la classe à l'origine de l'association et doit être unique par rapport aux attributs et aux autres noms de rôle.
`visibility`	public, private, protected, package.	Précise la visibilité de l'extrémité de l'association du point de vue de la classe de l'extrémité opposée. Public : Les classes peuvent naviguer à travers l'association en utilisant le nom de rôle dans les expressions de naviguation. Protected : les descendants de la classe source peuvent naviguer à travers l'association de la même manière que la visibilité public. Private : seulement la classe source peut naviguer à travers l'association. Package : les classes du même package et de ses sous-packages de n'importe quel niveau peuvent naviguer à travers l'association.
`ordering`	unordered, ordered, sorted, etc.	Précise si l'ensemble des liens est ordonné. Unordered : ensemble de liens avec l'absence d'un ordre inhérent. Ordered : un ensemble de liens qui peut être parcourt dans un ordre. Autres possibilités peuvent être définies par la déclaration de mots-clés additionnels en utilisant les mécanismes d'extension de UML.
`changeability`	changeable, frozen, addOnly	Précise si les liens peuvent être crées ou détruits après l'initialisation des objets de l'extrémité opposée de l'association. Changeable : aucune restriction sur la création et la destruction des liens. Frozen : aucun lien ne peut être créer ou détruit après l'initialisation des objets de l'extrémité opposée. AddOnly : aucun lien ne peut être détruit après l'initialisation des objets de l'extrémité opposée. Les liens peuvent être crées à tout moment.

TAB. 5 Description des caractéristiques de l'association et du rôle d'association en UML.

Caractéristique	Valeurs possibles	Signification
`multiplicity`	[1..1], [1..N], etc.	Une fois placé sur l'extrémité cible de l'association, elle précise le nombre d'instances cibles qui peuvent être associées à chaque instance de l'origine de cette association.
`aggregation`	aggregate, composite, none	Quand elle est mise sur l'extrémité cible de l'association, elle spécifie si la classe cible est une aggregation par rapport à la classe source. Seulement une seule extrémité peut être une aggregation. Aggregate : la classe cible est une agrégation. Par conséquent, la classe source est une partie est doit avoir la valeur de la propriété agrégation positionnée à 'none'. La partie peut participer à une autre agrégation. Composite : la classe cible est une composition. Par conséquent, la classe source est une partie est doit avoir la valeur de la propriété agrégation positionnée à 'none'. La partie est fortement liée par la classe composée et ne peut participer à une autre classe composée.
`targetScope`	instance, classifier	Spécifie si la cible est une instance ordinaire ou une classe. Instance : une valeur d'instance fait partie de chaque lien. Ceci est la l'état par défaut. Classifier : la classe elle même fait partie de chaque lien.
`isNaviguable`	true, false	Une fois mis sur l'extrémité opposée, il précise si le passage de l'instance source à l'instance cible correspondante est possible. True : l'association peut être naviguée par la classe source en utilisant le nom de rôle dans les expressions de navigation.

TAB. 6 Description des caractéristiques de l'association et du rôle d'association en UML (suite).

Annexe C

FCR du modèle Jetsmiles

Le contexte des classificateurs encode les classes du modèle de la Figure 8.2 suivant le schéma d'encodage décrit par la Table 1. Les objets formels sont les classes et les types de données employés avec les attributs et méthodes des classes. Les attributs formels sont les propriétés de la classe dans le méta-modèle UML. À noter que le contexte des classificateurs comporte peut d'attributs formel par contre beaucoup de relations avec les autres contexte de la FCR.

La Table 8 et la Table 9 représente le contexte des attributs issus de la Figure 8.2. Les objets formels sont les différents attributs de classes alors que les attributs formels représentent les propriétés décrites par les Tables 2 et 3.

Le contexte des associations de la Table 10 encode les associations du modèle de la Figure 8.2. Les objets formels sont les associations entre les différentes classes du modèle. Le seul attribut formel est le nom de l'association. Cet attribut pluri-valué est échantillonné de manière nominale.

La Table 11 représente le contexte des rôles qui encode les extrémités des associations du modèle de la Figure 8.2. Par conséquent, les objets formels sont les différents rôles dans le modèle et les attributs formels sont les propriétés d'un rôle telles que définies dans le méta-modèle UML et dont la signification est donnée par la Table 5 et la Table 6.

La Table 13 représente une relation dans la FCR du modèle de la Figure 8.2. En effet, elle associe à chaque attribut du modèle le type de la donnée qu'il représente. Ceci

	Classificateurs												
	Name.JCA	Visibility.public	isClass	Name.Passenger	Name.BookingPoints	Name.CourtesyPoints	Name.FlightPoints	Name.Rule	Name.String	Name.Integer	Name.Date	Name.Boolean	Name.Character
JCA	X	X	X										
Passenger		X	X	X									
BookingPoints		X	X		X								
CourtesyPoints		X	X			X							
FlightPoints		X	X				X						
Rule		X	X					X					
String		X	X						X				
Integer		X	X							X			
Date		X	X								X		
Boolean		X	X									X	
Character		X	X										X

TAB. 7 – Contexte binaire encodant les classificateurs de la Figure 8.2.

	Attributs												
	Name.firstname	Visibility.private	InitialValue.null	Multiplicity.[1,1]	Changeability.changeable	OwnerScope.instance	TargetScope.instance	Name.lastname	Name.title	Name.birthday	Name.gender	Name.redeemLock	Name.paxNumber
JCA.firstname	X	X	X	X	X	X	X						
JCA.lastname		X	X	X	X	X	X	X					
JCA.title		X	X	X	X	X	X		X				
JCA.birthday		X	X	X	X	X	X			X			
JCA.gender		X	X	X	X	X	X				X		
JCA.redeemLock		X	X	X	X	X	X					X	
Passenger.paxNumber		X	X	X	X	X	X						X
Passenger.firstname	X	X	X	X	X	X	X						
Passenger.lastname		X	X	X	X	X	X	X					
Passenger.title		X	X	X	X	X	X		X				
Passenger.typeCode		X	X	X	X	X	X						
Passenger.ticketNumber		X	X	X	X	X	X						
Passenger.sex		X	X	X	X	X	X						
BookingPoints.points		X	X	X	X	X	X						
BookingPoints.added		X	X	X	X	X	X						
BookingPoints.modified		X	X	X	X	X	X						
BookingPoints.valid		X	X	X	X	X	X						
BookingPoints.validDate		X	X	X	X	X	X						
CourtesyPoints.points		X	X	X	X	X	X						
CourtesyPoints.added		X	X	X	X	X	X						
CourtesyPoints.modified		X	X	X	X	X	X						
CourtesyPoints.awardedBy		X	X	X	X	X	X						
CourtesyPoints.validDate		X	X	X	X	X	X						
CourtesyPoints.comment		X	X	X	X	X	X						
FlightPoints.points		X	X	X	X	X	X						
FlightPoints.added		X	X	X	X	X	X						
FlightPoints.modified		X	X	X	X	X	X						
FlightPoints.valid		X	X	X	X	X	X						
FlightPoints.validDate		X	X	X	X	X	X						
Rule.ruleName		X	X	X	X	X	X						
Rule.description		X	X	X	X	X	X						

TAB. 8 – Contexte binaire des attributs des classes de la Figure 8.2.

Attributs - suite												
	Name.typeCode	Name.ticketNumber	Name.sex	Name.points	Name.added	Name.modified	Name.valid	Name.validDate	Name.awardedBy	Name.comment	Name.ruleName	Name.description
JCA.firstname												
JCA.lastname												
JCA.title												
JCA.birthday												
JCA.gender												
JCA.redeemLock												
Passenger.paxNumber												
Passenger.firstname												
Passenger.lastname												
Passenger.title												
Passenger.typeCode	X											
Passenger.ticketNumber		X										
Passenger.sex			X									
BookingPoints.points				X								
BookingPoints.added					X							
BookingPoints.modified						X						
BookingPoints.valid							X					
BookingPoints.validDate								X				
CourtesyPoints.points				X								
CourtesyPoints.added					X							
CourtesyPoints.modified						X						
CourtesyPoints.awardedBy									X			
CourtesyPoints.validDate								X				
CourtesyPoints.comment										X		
FlightPoints.points				X								
FlightPoints.added					X							
FlightPoints.modified						X						
FlightPoints.valid							X					
FlightPoints.validDate								X				
Rule.ruleName											X	
Rule.description												X

TAB. 9 – Suite du contexte binaire des attributs des classes de la Figure 8.2.

	Associations								
	Name.acctocp	Name.acctofp	Name.acctobp	Name.passtoacc	Name.fptopass	Name.bptopass	Name.cptorule	Name.fptorule	Name.bptorule
acctocp	X								
acctofp		X							
acctobp			X						
passtoacc				X					
fptopass					X				
bptopass						X			
cptorule							X		
fptorule								X	
bptorule									X

TAB. 10 – Contexte binaire des associations de la Figure 8.2.

définie une relation inter-objets, appelée `type`, entre les attributs et les classificateurs.

Rôles des associations								
	Name.account	Visibility.public	Multiplicity.[1,1]	Ordering.unordered	Aggregation.composite	Changeability.changeable	TargetScope.instance	Name.courtesyPoints
acctocp.account	X	X	X	X	X	X	X	
acctocp.courtesyPoints		X		X		X	X	X
acctofp.account	X	X	X	X	X	X	X	
acctofp.flightPoints		X		X		X	X	
acctobp.bookingPoints		X		X	X	X	X	X
acctobp.account	X	X		X		X	X	
passtoacc.account	X	X	X	X		X	X	
passtoacc.passenger		X		X		X	X	
fptopass.flightPoints		X		X		X	X	
fptopass.passenger		X	X	X		X	X	
bptopass.bookingPoints		X		X		X	X	
bptopass.passenger		X	X	X		X	X	
cptorule.courtesyPoints		X		X		X	X	X
cptorule.rule		X	X	X		X	X	
fptorule.flightPoints		X		X		X	X	
fptorule.rule		X	X	X		X	X	
bptorule.bookingPoints		X		X		X	X	
bptorule.rule		X	X	X		X	X	

TAB. 11 – Contexte binaire des rôles d'associations de la Figure 8.2.

Rôles des associations - Suite							
	Multiplicity.[0,n]	Aggregation.none	isNavigable	Name.flightPoints	Name.bookingPoints	Name.passenger	Name.rule
acctocp.account							
acctocp.courtesyPoints	X	X	X				
acctofp.account							
acctofp.flightPoints	X	X	X	X			
acctobp.bookingPoints	X				X		
acctobp.account	X	X	X				
passtoacc.account		X	X				
passtoacc.passenger	X	X				X	
fptopass.flightPoints	X	X		X			
fptopass.passenger		X	X			X	
bptopass.bookingPoints	X	X			X		
bptopass.passenger		X	X			X	
cptorule.courtesyPoints	X	X					
cptorule.rule		X	X				X
fptorule.flightPoints	X	X		X			
fptorule.rule		X	X				X
bptorule.bookingPoints	X	X			X		
bptorule.rule		X	X				X

TAB. 12 – Suite du contexte binaire des rôles d'associations de la Figure 8.2.

type : attributs x classificateurs					
	String	Date	Character	Boolean	Integer
JCA.firstname	X				
JCA.lastname	X				
JCA.title	X				
JCA.birthday		X			
JCA.gender			X		
JCA.redeemLock				X	
Passenger.paxNumber	X				
Passenger.firstname	X				
assenger.lastname	X				
Passenger.title	X				
Passenger.typeCode	X				
Passenger.ticketNumber	X				
Passenger.sex			X		
BookingPoints.points					X
BookingPoints.added		X			
BookingPoints.modified		X			
BookingPoints.valid				X	
BookingPoints.validDate		X			
CourtesyPoints.points					X
CourtesyPoints.added		X			
CourtesyPoints.modified		X			
CourtesyPoints.awardedBy	X				
CourtesyPoints.validDate		X			
CourtesyPoints.comment	X				
FlightPoints.points					X
FlightPoints.added		X			
FlightPoints.modified		X			
FlightPoints.valid				X	
FlightPoints.validDate		X			
Rule.ruleName	X				
Rule.description	X				

TAB. 13 – Relation inter-objets type entre les attributs et les classificateurs de la Figure 8.2.

owned_attribute : classificateurs x attributs																
	JCA.firstname	JCA.lastname	JCA.title	JCA.birthday	JCA.gender	JCA.redeemLock	Passenger.paxNumber	Passenger.firstname	Passenger.lastname	Passenger.title	Passenger.typeCode	Passenger.ticketNumber	Passenger.sex	BookingPoints.points	BookingPoints.added	BookingPoints.modified
JCA	X	X	X	X	X	X										
Passenger							X	X	X	X	X	X	X			
BookingPoints														X	X	X

TAB. 14 – Relation inter-objets `owned_attribute` entre les classificateurs et les attributs de la Figure 8.2.

owned_attribute : classificateurs x attributs - Suite															
	BookingPoints.valid	BookingPoints.validDate	CourtesyPoints.points	CourtesyPoints.added	CourtesyPoints.modified	CourtesyPoints.awardedBy	CourtesyPoints.validDate	CourtesyPoints.comment	FlightPoints.points	FlightPoints.added	FlightPoints.modified	FlightPoints.valid	FlightPoints.validDate	Rule.ruleName	Rule.description
BookingPoints	X	X													
CourtesyPoints			X	X	X	X	X	X							
FlightPoints									X	X	X	X	X		
Rule														X	X

TAB. 15 – Suite de la relation inter-objets `owned_attribute` entre les classificateurs et les attributs de la Figure 8.2.

owned_operation : classificateurs x operations		
	JCA.getName	Passenger.getName
JCA	X	
Passenger		X

TAB. 16 – Relation inter-objets owned_operation entre les classificateurs et les méthodes de la Figure 8.2.

owned_association_end : classificateurs x rôles des associations																		
	acctocp.account	acctocp.courtesyPoints	acctofp.account	acctofp.flightPoints	acctobp.bookingPoints	acctobp.account	passtoacc.account	passtoacc.passenger	fptopass.flightPoints	fptopass.passenger	bptopass.bookingPoints	bptopass.passenger	cptorule.courtesyPoints	cptorule.rule	fptorule.flightPoints	fptorule.rule	bptorule.bookingPoints	bptorule.rule
JCA	X		X			X	X											
Passenger								X		X		X						
BookingPoints					X						X						X	
CourtesyPoints		X											X					
FlightPoints				X					X						X			
Rule														X		X		X

TAB. 17 – Relation inter-objets owned_association_end entre les classificateurs et les rôles des associations de la Figure 8.2.

return_type : opérations x classificateurs	
	String
JCA.getName	X
Passenger.getName	X

TAB. 18 – Relation inter-objets return_type entre les les méthodes des classes et les classificateurs de la Figure 8.2.

belong_to_association : rôles x associations									
	acctocp	acctofp	acctobp	passtoacc	fptopass	bptopass	cptorule	fptorule	bptorule
acctocp.account	X								
acctocp.courtesyPoints	X								
acctofp.account		X							
acctofp.flightPoints		X							
acctobp.bookingPoints			X						
acctobp.account			X						
passtoacc.account				X					
passtoacc.passenger				X					
fptopass.flightPoints					X				
fptopass.passenger					X				
bptopass.bookingPoints						X			
bptopass.passenger						X			
cptorule.courtesyPoints							X		
cptorule.rule							X		
fptorule.flightPoints								X	
fptorule.rule								X	
bptorule.bookingPoints									X
bptorule.rule									X

TAB. 19 – Relation inter-objets `belong_to_association` entre les méthodes des classes et les classificateurs de la Figure 8.2.

first_end : associations x rôles									
	acctocp.account	acctofp.account	acctobp.bookingPoints	passtoacc.account	fptopass.flightPoints	bptopass.bookingPoints	cptorule.courtesyPoints	fptorule.flightPoints	bptorule.bookingPoints
acctocp	X								
acctofp		X							
acctobp			X						
passtoacc				X					
fptopass					X				
bptopass						X			
cptorule							X		
fptorule								X	
bptorule									X

TAB. 20 – Relation inter-objets `first_end` entre associations et rôles de la Figure 8.2.

second_end : associations x rôles									
	acctocp.courtesyPoints	acctofp.flightPoints	acctobp.account	passtoacc.passenger	fptopass.passenger	bptopass.passenger	cptorule.rule	fptorule.rule	bptorule.rule
acctocp	X								
acctofp		X							
acctobp			X						
passtoacc				X					
fptopass					X				
bptopass						X			
cptorule							X		
fptorule								X	
bptorule									X

TAB. 21 – Relation inter-objets second_end entre associations et rôles de la Figure 8.2.

specified_class : rôles des associations x classificateurs						
	JCA	Passenger	BookingPoints	CourtesyPoints	FlightPoints	Rule
acctocp.account	X					
acctocp.courtesyPoints				X		
acctofp.account	X					
acctofp.flightPoints					X	
acctobp.bookingPoints			X			
acctobp.account	X					
passtoacc.account	X					
passtoacc.passenger		X				
fptopass.flightPoints					X	
fptopass.passenger		X				
bptopass.bookingPoints			X			
bptopass.passenger		X				
cptorule.courtesyPoints				X		
cptorule.rule						X
fptorule.flightPoints					X	
fptorule.rule						X
bptorule.bookingPoints			X			
bptorule.rule						X

TAB. 22 – Relation inter-objets `specified_class` entre rôles d'associations et classificateurs de la Figure 8.2.