Institutional Repository - Research Portal
Dépôt Institutionnel - Portail de la Recherche
researchportal.unamur.be
University of Namur

THESIS / THÈSE

MASTER EN SCIENCES INFORMATIQUES

Système de validation intelligente de règles comptables avec l'aide de la programmation par contraites

Thirifay, Olivier

Award date:
2002

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 04. May. 2026

FACULTÉS UNIVERSITAIRES NOTRE-DAME DE LA PAIX
INSTITUT D'INFORMATIQUE
NAMUR

Système de validation intelligente de règles comptables avec l'aide de la programmation par contraintes

Olivier Thirifay

Mémoire présenté en vue de l'obtention
du grade de
MAÎTRE EN INFORMATIQUE

ANNÉE ACADÉMIQUE 2001-2002

Résumé

Ce travail abordera la possibilité de valider des données saisies par un utilisateur dans le domaine de la comptabilité d'une entreprise. Ces validations s'appliqueront sur des règles exprimées par les comptables et permettant d'effectuer des regroupements d'informations. Elles permettront de détecter des erreurs avant que leurs propagations dans le processus de création de documents comptables ne les rendent difficiles à localiser. Le problème posé se prêtant bien à la modélisation logique, nous aborderons la programmation logique et plus particulièrement la programmation logique avec contraintes. Celle-ci étant encore peu utilisée en entreprise, l'occasion nous est donnée de montrer l'avantage que peut retirer une entreprise d'utiliser ce style de programmation.

Mots-clés : programmation logique, modélisation, programmation logique avec contraintes, validation de données, comptabilité.

Abstract

This work will tackle the possibility of validating data captured by a user in the field of the accountancy of a company. These validations will be performed on rules expressed by the accountants and achieving regroupings of informations. This will make it possible to detect errors before their propagations in the process of reporting make them difficult to locate. The problem formulated lends itself well to logical modeling. The work deals with the logical programming and more particularly with the constraint logic programming. As constraint logic programming is still being used little in companies, the work shows the advantage a company can draw from this style of programming.

Keywords : logic programming, modelization, constraints logic programming, data validation, accounting.

Avant-propos

Je tiens à remercier Monsieur Vincent Englebert pour ses conseils et remarques apportées tout au long de la rédaction de ce mémoire.

Je voudrais aussi exprimer ma reconnaissance aux différentes personnes que j'ai pu rencontrer durant mon stage à la banque Dexia-BIL et tout particulièrement Messieurs Thierry Lamouline et Patrick Boillot pour leurs explications ayant permis le bon déroulement de ce stage.

Mes remerciements s'adressent également à tous ceux qui, de près ou de loin, ont également contribué à la rédaction de ce mémoire, de par leur soutien, leurs remarques et leurs innombrables relectures.

Table des matières

Glossaire **v**

Introduction **1**

1 Contexte **3**

1.1 Dexia Banque Internationale à Luxembourg 3

1.2 Le reporting bancaire 4

1.2.1 La comptabilité 4

1.2.2 Les autorités de contrôles 7

1.2.3 Les documents à produire 8

2 Étude de l'existant **11**

2.1 Introduction 11

2.2 Le reporting sur mainframe 11

2.3 Critique de l'existant 13

3 Le "nouveau reporting" **17**

3.1 Introduction 17

3.2 Les principes 17

3.3 L'architecture globale et ses composants 20

3.3.1 L'éditeur de règles 26

3.3.2 La validation intelligente des règles 29

4 Spécifications du problème de validation intelligente de règles comptables **31**

4.1 Introduction 31
4.2 Description des règles de regroupement 32
4.2.1 Quelques conventions de la grammaire *BNF* 32
4.2.2 Les règles de regroupement 32
4.3 La compatibilité 39
4.3.1 Besoins 39
4.3.2 Définition 40
4.3.3 Formalisation mathématique 41
4.4 Le recouvrement 42
4.4.1 Besoin 42
4.4.2 Définition 45
4.4.3 Formalisation mathématique 45
4.5 La définition complète 46
4.5.1 Besoin 46
4.5.2 Définition 47
4.5.3 Formalisation mathématique 49
4.6 Le complémentaire 49
4.6.1 Besoin 49
4.6.2 Définition 49
4.7 Conclusion 51

5 La programmation logique avec contraintes **53**

5.1 Introduction 53
5.2 La logique du premier ordre 53
5.3 La programmation logique 57
5.4 La programmation logique avec contraintes 65
5.4.1 Complexité des problèmes 66

5.4.2 Comparaison à l'aide d'un exemple : SEND+MORE=MONEY 68
5.5 Le langage ECL^iPS^e 72
5.5.1 Introduction 72
5.5.2 Caractéristiques du langage 73
5.5.3 Les librairies disponibles 73
5.5.4 Fonctionnement 75
5.5.5 La communication entre ECL^iPS^e et d'autres langages 77
5.6 Méthodes de résolution 78

6 Application de la programmation avec contraintes à notre problème 79
6.1 Introduction 79
6.2 Architecture du prototype 79
6.3 Modélisation des règles de regroupement 82
6.4 Modélisation des besoins identifiés 83
6.4.1 La compatibilité 83
6.4.2 Le recouvrement 85
6.4.3 La définition complète 89
6.5 Résultats obtenus 90
6.5.1 La compatibilité 91
6.5.2 Le recouvrement 92
6.5.3 La définition complète 92

Conclusion 95

Bibliographie 97

Annexes 99

Glossaire

BNF : `Backus Naur Form`. Grammaire permettant de définir formellement un langage.

SQL : `Structured Query Language`. Langage permettant d'effectuer des requêtes sur une base de données.

XML : `Extensible Markup Language`. Format de données principalement prévu pour les échanges de données sur Internet.

CLP : `Constraint Logic Programming`, programmation logique avec contraintes.

CSP : `Constraint Satisfaction Problem`, problème de satisfaction de contraintes.

Reporting : Processus de production des documents comptables établissant un compte rendu de la situation financière de l'entreprise.

Règles de regroupement : Règles exprimées par les comptables afin de réaliser des aggrégations d'informations dans le but de produire des documents comptables tels les comptes annuels.

Introduction

De nos jours, de par la mondialisation de l'économie et le développement des bourses, les informations comptables des entreprises sont des données très prisées. Les clients, les actionnaires, les pouvoirs publics, ... recherchent ces informations pour pouvoir évaluer la qualité d'une entreprise et décider de traiter ou non avec elle. Toutefois, nous ne sommes pas à l'abri des abus. Une entreprise pourrait embellir ses informations notamment pour attirer des partenaires. C'est pourquoi les législateurs, dans de nombreux pays, ont décidé d'imposer aux entreprises des obligations concernant la tenue de leurs comptes et la publication de ceux-ci ainsi que d'instaurer des contrôles. L'objectif est que chacun puisse se faire une idée la plus réaliste possible de la situation financière d'une entreprise à l'aide de ces informations. Les entreprises sont désormais obligées d'observer des conventions standardisant la présentation des informations. Une entreprise disposant d'un nombre considérable de données et les conventions étant nombreuses, il a vite fallu informatiser ce processus de production. Mais toutes les données présentes dans le système d'informations de l'entreprise ne sont pas nécessaires pour cette présentation. Un tri est donc effectué grâce à des règles exprimées par les comptables et qui s'appliquent sur les données. Ces règles doivent évidemment tenir compte des conventions édictées par la loi et par divers organismes de contrôles. L'erreur étant humaine, certaines se glisseront parmi ces règles, faussant ainsi le résultat de ce processus.

Ce travail va s'attacher à montrer les avantages de la programmation logique avec contraintes afin de réaliser la validation des règles exprimées par les comptables.

Nous aborderons dans le premier chapitre le contexte du stage qui s'est déroulé à la banque Dexia-BIL à Luxembourg. Ensuite, nous décrirons dans le deuxième chapitre l'existant et analyserons les raisons d'introduire un nouveau système. Dans le troisième chapitre, nous détaillerons le nouveau système que l'on désire mettre en place. A partir du quatrième chapitre, nous nous focaliserons sur le problème particulier qui nous concerne, à savoir la validation intelligente de données. Celle-ci constitue un sous-projet du système à mettre en place. Nous spécifierons le problème dans le chapitre 4. Pour pouvoir montrer l'intérêt de la programmation logique avec contraintes que nous utiliserons pour résoudre ce problème, nous introduirons la programmation logique et la programmation logique avec contraintes dans le cinquième chapitre. Ce sera également l'occasion de présenter brièvement le langage utilisé afin de réaliser le prototype, ECL^iPS^e. Le sixième chapitre sera consacré à la modélisation du problème que nous aurons posé et aux résultats obtenus suite aux tests effectués. Le moment sera alors venu de tirer les conclusions de cette expérience.

Chapitre 1

Contexte

1.1 Dexia Banque Internationale à Luxembourg

FIG. 1.1: Logo du groupe Dexia.

Depuis sa fondation en 1856, Dexia Banque Internationale à Luxembourg (Dexia-BIL) a joué un rôle actif dans le développement des principales phases de l'économie luxembourgeoise. En 1991, le Crédit Communal de Belgique entre dans le capital de Dexia-BIL et devient actionnaire majoritaire. Depuis 1996, suite à l'alliance entre le Crédit Communal de Belgique et le Crédit Local de France, Dexia-BIL fait partie du groupe Dexia, actif dans les trois métiers suivants :

- le crédit aux autorités locales ;
- la banque commerciale et la banque privée ;
- la gestion d'actifs et l'administration de fonds.

Depuis sa naissance, le groupe Dexia n'a eu de cesse de croître via une série d'acquisitions sur ses marchés domestiques que sont la France, la Belgique et le Luxembourg, mais aussi de plus en plus dans les autres pays européens afin de développer ses trois principaux métiers. Fin 2001, le groupe emploie près de 26.141 personnes dans une vingtaine de pays et figure parmi les 25 premières banques de l'Union Européenne.
Aujourd'hui, Dexia-BIL exerce le métier de banque commerciale, de banque privée ainsi que d'administration de fonds d'investissement à Luxembourg. Elle tente également de développer ces deux dernières activités sur le plan international. Pour ce faire, elle emploie 2.716 employés à Luxembourg fin de l'année 2001 et près de 5.900 employés répartis dans 22 pays parmi lesquels la Suisse, l'Allemagne, Singapour, Monaco, les îles Cayman, l'Angleterre, Hong Kong et le Japon.

1.2 Le reporting bancaire

L'objectif du stage s'intègre dans un projet appelé le "nouveau reporting". Ce projet et le système existant, que nous détaillerons tous deux par après, ont pour objectif de faciliter une des tâches de la comptabilité à savoir réaliser la synthèse des informations comptables. Avant d'aller plus loin, nous allons d'abord décrire brièvement ce qu'est la comptabilité et, en même temps, ce qu'est le reporting.
Pour de plus amples informations concernant la comptabilité d'une entreprise, le lecteur est invité à se reporter à [12].

1.2.1 La comptabilité

Une entreprise est une organisation dans laquelle des facteurs de production contribuent conjointement à créer une valeur ajoutée qui les rémunère. L'activité de l'entreprise peut être sommairement illustrée par la figure 1.2 représentant la transformation continue de biens et services en monnaie. Cette activité entraîne quotidiennement l'entreprise à effectuer des transactions, qu'elle enregistre notamment pour sa gestion quotidienne. Cet enregistrement des opérations est ce que l'on nomme la comptabilité.

Celle-ci permet d'établir l'image financière de l'entreprise, d'enregistrer la variation de sa richesse.

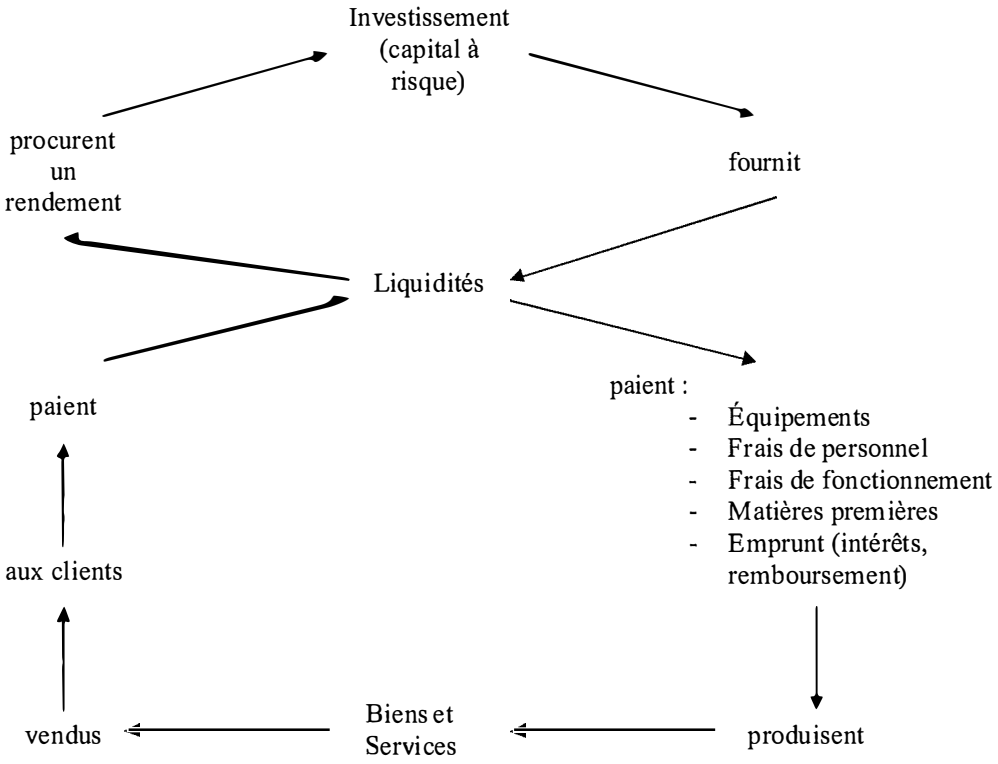


FIG. 1.2: Le cycle financier de l'entreprise.

Chaque entreprise entretient des relations avec les partenaires que sont ses clients, ses fournisseurs, son personnel, les pouvoirs publics et ses bailleurs de fonds. Chacun possède dans celle-ci des intérêts, parfois conflictuels et qui sont résolus par l'établissement de relations contractuelles. C'est dans ce contexte que le besoin en informations comptables se fait ressentir afin de pouvoir vérifier le respect des engagements de chacun. La comptabilité se révèle être un outil de surveillance réciproque. Ces informations doivent donc être publiées pour que tout le monde puisse en disposer. De plus, les informations comptables sont aujourd'hui recherchées, analysées

et interprétées par un grand nombre d'intervenants. Il est donc primordial que ces informations reflètent la situation financière de l'entreprise le plus fidèlement possible. Cette fidélité, la législation essaie de la garantir en imposant diverses obligations aux entreprises comme, par exemple, la publicité de leurs informations mais aussi en instaurant des contrôles. Toutefois, les conventions en matière comptable sont nombreuses et leurs interprétations ainsi que leurs mise en oeuvre peuvent être très diverses. Chaque pays possède formule ses exigences au travers de lois, instaurant ainsi un standard. Ces exigences évoluent dans le temps, pour répondre à de nouvelles possibilités ou pour éviter des pratiques abusives. L'entreprise doit donc pouvoir s'adapter facilement à ces exigences.

Deux approches sont à distinguer dans le système d'information comptable de l'entreprise en fonction des besoins des utilisateurs de l'information produite :

- La comptabilité financière qui s'attache à rendre compte du patrimoine de l'entreprise, de ses engagements financiers à l'égard des tiers et du résultat (bénéfice ou perte). Ces informations sont surtout recherchées par des partenaires non gestionnaires de l'entreprise comme le personnel, les actionnaires créanciers, les pouvoirs publics, ...

- La comptabilité de gestion ou comptabilité analytique fournit des informations plus détaillées, parfois confidentielles, à l'usage des gestionnaires de l'entreprise.

La différence entre ces deux approches se situe au niveau du classement des charges et produits, ce qui signifie que la ventilation des données sera différente. En effet la première approche les classe "par nature" : revenus des ventes, frais de personnel, ... tandis que la seconde les classe "par destination" : ventes du bien A, ventes du bien B, frais de publicité encourus pour promouvoir le bien A, ... Pour réaliser la ventilation des données de l'entreprise suivant les destinataires de l'information, les comptables exprimeront des règles qui s'appliqueront sur ces données. Toutes ces informations comptables seront alors consignées dans des documents spécifiques.

1.2.2 Les autorités de contrôles

Afin de garantir la fidélité ainsi que la publicité des informations comptables fournies par les entreprises, le législateur a imposé certaines obligations aux entreprises et des institutions se sont vues attribuer des missions de surveillance et de collectes des informations. C'est notamment le cas, en ce qui concerne le Luxembourg, des institutions suivantes :

- la **C**ommission de **S**urveillance du **S**ecteur **F**inancier (CSSF),
- la **B**anque **C**entrale du **L**uxembourg (BCL),
- l'**I**nstitut **B**elgo-**L**uxembourgeois de **C**hange (IBLC).

La Commission de Surveillance du Secteur Financier (CSSF)

La Commission de Surveillance du Secteur Financier a repris les compétences exercées précédemment par l'Institut Monétaire Luxembourgeois (IML) devenu depuis Banque Centrale du Luxembourg (BCL). Elle veille à l'application et au respect des lois et réglementations concernant le secteur financier. Elle peut émettre des circulaires précisant les modalités d'application de différentes dispositions légales concernant les entités surveillées, publier des règles spécifiques à certains domaines d'activité et émettre des recommandations relatives à l'exercice des activités du secteur financier. La CSSF exerce donc une surveillance envers les entreprises du secteur financier dont les objectifs sont les suivants :

- promouvoir une politique d'affaires réfléchie et prudente, conforme aux exigences réglementaires ;
- protéger la stabilité financière des entreprises surveillées et du secteur financier dans son ensemble ;
- veiller à la qualité de l'organisation et des systèmes de contrôle interne ;
- renforcer la qualité de la gestion des risques.

La Banque Centrale du Luxembourg (BCL)

La Banque centrale du Luxembourg est chargée, entre autres missions, de participer à la surveillance du système financier en vue d'assurer la stabilité du système financier luxembourgeois.

L'Institut belgo-luxembourgeois du change (IBLC)

Cet institut n'exerce plus que des missions statistiques depuis que le double marché des changes a été supprimé le 5 mars 1990 en Belgique et au Grand-duché de Luxembourg.

Les entreprises sont tenues de remettre périodiquement à ces diverses institutions des documents financiers synthétisant toutes leurs informations comptables. Ces documents doivent répondre à un certain nombre de conditions de forme comme la présentation, la dactylographie ou la qualité et les dimensions du papier. Ils peuvent également être déposés sous forme électronique.

1.2.3 Les documents à produire

La banque Dexia-BIL, comme toute autre société, a l'obligation de transmettre sur une base périodique des informations financières sur ses activités aux institutions de contrôle. Les informations qui permettent la gestion quotidienne ne suffisent pas aux institutions de contrôles. Elles exigent des informations plus détaillées suivant un système de classifications. Par exemple, les informations peuvent être classées selon le type de client (personne physique, personne morale, établissement de crédit, ...), selon le pays de résidence du client, ... Cette synthèse des informations sera rapportée à travers divers documents financiers et notamment les comptes annuels. Ces derniers sont subdivisés en quatre parties principales :

- le **bilan**, qui donne un relevé des avoirs et des engagements d'une entreprise. Il traduit la situation patrimoniale de celle-ci à la fin d'une période donnée, qualifiée d'"exercice comptable". Cette période correspond fréquemment, mais pas nécessairement, à l'exercice civil. Le bilan est composé de deux parties. D'un côté, il y a les biens de l'entreprise ou "actifs" et de l'autre se retrouvent les sources de financement de l'entreprise ou "passifs". Par exemple, font partie des actifs, les immeubles dont elle est propriétaire, son parc de machines, ses stocks, les créances sur la clientèle, etc. Certains avoirs immatériels peuvent également avoir une valeur patrimoniale et se retrouver à l'actif du bilan comme par exemple les frais de recherche et développement, les brevets. Sont considérés comme des passifs, le capital investi par les actionnaires, les dettes commerciales, mais aussi les réserves constituées en vue de faire face à des dépenses attendues. Au bilan, tous ces actifs et ces passifs sont classés en rubriques précises ;

- le **compte de résultats** qui donne une vue d'ensemble des flux de recettes et de dépenses durant un exercice comptable. Les dépenses comprennent par exemple les frais de personnel ou les achats de matières premières. Les recettes se composent entre autres du chiffre d'affaires, des intérêts dégagés par les actifs financiers ou de plus-values réalisées lors de la vente d'un immeuble ;

- les **annexes**, dans lesquelles diverses rubriques du bilan et du compte de résultats sont ventilées en détail et expliquées de manière plus approfondie ;

- le **bilan social**, lequel contient des informations spécifiques relatives à l'emploi dans l'entreprise parmi lesquelles le nombre de personnes occupées, la rotation du personnel ou les formations suivies par celui-ci.

Les documents financiers doivent permettre à quiconque de se faire une idée sur la situation financière d'une entreprise à un moment donné. Ils permettent donc d'établir

un compte rendu des activités et de la situation financière de l'entreprise à ce moment. Le processus de production de ces documents est généralement désigné par le terme "reporting", venant de l'anglais "to report", à savoir "rapporter, faire un compte rendu". Le lecteur pourra trouver en annexe A des explications sur les ventilations à réaliser ainsi qu'un exemple de documents à remettre en annexe B.

Chapitre 2

Étude de l'existant

2.1 Introduction

Nous avons vu que l'entreprise dispose dans son système d'informations de données relatives à ses transactions. Celles-ci sont nécessaires pour produire les documents requis par les divers partenaires de l'entreprise. Nous avons aussi vu que chacun formule des exigences particulières concernant le contenu, voire la présentation, des documents et que certaines informations sont destinées uniquement aux dirigeants de l'entreprise pour cause de confidentialité. Toutes ces raisons ainsi que le nombre colossal d'informations à traiter ont rendu nécessaire la mise en place d'un système permettant d'automatiser cette tâche. Nous allons maintenant examiner le système informatique mis en place par la banque Dexia-BIL pour ce faire.

Dans ce chapitre, nous verrons d'abord le système existant et analyserons ensuite ses principaux inconvénients. Cette analyse nous permettra de voir les améliorations à apporter à cet existant afin de disposer d'un système plus efficace.

2.2 Le reporting sur mainframe

L'informatique actuelle de la banque Dexia-BIL est assuré par le BIL Lux System (BLS). A l'origine, le BLS est un système intégré suisse qui a été fortement adapté

pour satisfaire aux besoins spécifiques de Dexia-BIL et aux exigences luxembourgeoises. Le système de reporting actuel a été mis en en place en 1994. Il fait partie intégrante du système BLS et tourne sur un mainframe. Un mainframe désigne un gros ordinateur auquel un ou plusieurs ordinateurs et/ou terminaux sont connectés pour partager ses ressources et sa puissance de calcul. La figure 2.1 décrit le système existant que nous allons détailler.

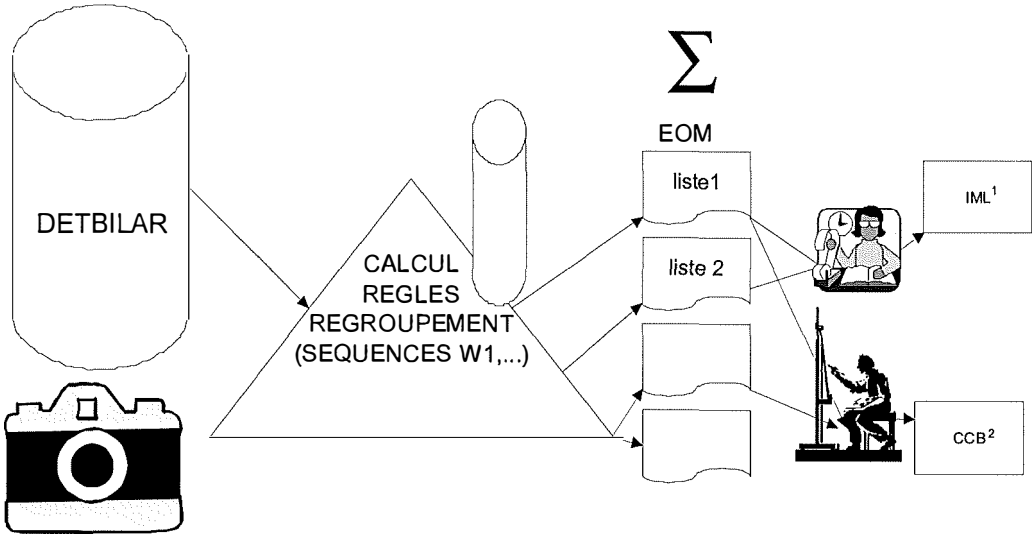


FIG. 2.1: Le reporting sous mainframe.

Le BLS s'appuie sur une base de données contenant toutes les informations gérées électroniquement par la banque. Cette base de données, de par la diversité et la quantité d'informations qu'elle brasse, est très volumineuse et nécessite une vigilance importante pour le maintient de ses performances. Le système de reporting ne peut, par définition, travailler sur une base d'informations en évolution. Aussi, tous les

[1]l'Institut Monétaire Luxembourgeois (IML) est le prédécesseur de la Banque centrale du Luxembourg

[2]le Crédit Communal de Belgique (CCB) était la maison-mère de la banque BIL, renommée par après Dexia-BIL, lors de la mise en place de ce système.

mois, un backup de cette base de données est réalisé. On dispose ainsi d'une "photo" du système d'informations. De ce backup, les informations nécessaires à la production des divers documents financiers seront extraites et regroupées dans un fichier spécifique nommé `Detbilar`. Ces données récoltées, il faut encore réaliser la ventilation selon le système de classifications édicté par la législation et, de cette manière, le formatage des documents. Pour ce faire, il faut traduire les conventions édictées par la législation et les institutions de contrôle en règles de regroupement applicables sur les données informatisées. Ces règles sont définies dans le programme par l'intermédiaire de tables de paramétrages et s'appliqueront sur le contenu du fichier `Detbilar`. La ventilation des informations réalisée, le formatage des documents sera réalisé automatiquement et ils seront imprimés sous forme de listings. Ceux-ci feront alors l'objet de vérifications et, éventuellement, de corrections avant d'être envoyés à la maison-mère et aux institutions de contrôles.

2.3 Critique de l'existant

Nous allons voir ici quelles sont les améliorations susceptibles d'être apportées au système existant et pourquoi elles sont utiles.
On l'aura compris, ce système permet de réaliser un travail devenu considérable pour les comptables étant donné la quantité d'informations à traiter et de conventions à respecter. Un facteur de complexité supplémentaire est encore introduit puisque la ventilation des informations dépend aussi des destinataires de l'information. Mais ce travail automatisé n'empêche pas les erreurs d'apparaître dans les documents produits. Des contrôles sont donc effectués par le personnel afin de les débusquer et d'en trouver les causes pour les corriger. Il existe deux causes possibles pour une erreur : soit il s'agit d'une erreur de traduction des conventions, soit de l'existence de données incorrectes dans le système d'informations. Cette recherche peut se révéler longue et fastidieuse, d'autant plus que les corrections doivent être apportées manuellement

dans tous les documents. En effet, la production des documents financiers est un processus très coûteux en ressources et qui demande du temps (toute une nuit). Il n'est donc pas possible de l'exécuter après chaque modifications de règles ou de données. C'est pourquoi le système existant n'offre pas la possibilité de corriger les informations incorrectes dans le système, ni d'effectuer des contrôles sur la saisie des règles. Cette première fonctionnalité permettrait, après corrections, de reproduire des documents plus propres, c'est-à-dire ne devant pas faire l'objet de corrections manuelles a posteriori.

Une autre difficulté provient de la manière dont sont codées les traductions des conventions, c'est-à-dire les règles de regroupement. Celles-ci sont codées via des tables de paramétrages du système et font donc partie intégrante de celui-ci. Toute modification de ces règles nécessite donc l'intervention d'informaticiens. Or, ces demandes de modifications peuvent survenir fréquemment soit pour cause de corrections soit parce que les exigences en matière d'informations ont changé. En effet, ces exigences tant légales qu'à l'intérieur du groupe Dexia évoluent dans le temps. Pour continuer à satisfaire aux exigences réglementaires, les programmes ont été progressivement adaptés. Mais le cumul de ces adaptations ont petit à petit allongé la durée des traitements et complexifié les programmes. C'est pourquoi, ce système permettait de satisfaire les exigences définies lors de sa mise en place mais plus nécessairement les exigences actuelles. Par ailleurs, cette évolution des exigences nécessite de renseigner des informations qui ne l'étaient pas auparavant, ce qui exige d'adapter la base de données centrale et de répercuter les changements jusqu'au fichier `Detbilar`, ce qui devient de plus en plus difficile à réaliser.

On le voit, ce système est largement automatisé et n'offre donc pas beaucoup de liberté aux comptables. Ceux-ci ne peuvent pas agir sur le déroulement du processus de production sauf par l'intermédiaire des informaticiens. Les informaticiens, eux, ne disposent pas d'un outil de saisie des règles exprimées par les comptables. Cet absence se traduit par un manque de convivialité afin d'introduire les règles dans le système. Par exemple, dans le système d'informations de la banque, chaque pays est

identifié par un code numérique. Ces codes sont utilisés dans l'expression des règles, augmentant ainsi les risques d'erreurs. L'utilisation d'un libellé, par exemple le nom du pays, permettrait de réduire ce risque.

En conclusion, nous dirons que le système existant permet d'alléger la charge de travail des comptables mais ne semble plus adapté au travail qu'on voudrait lui attribuer aujourd'hui. Une refonte de ce système s'impose donc afin de permettre aux comptables d'influencer le processus de production de manière plus active, notamment en introduisant eux-mêmes les règles dans le programme.

Chapitre 3

Le "nouveau reporting"

3.1 Introduction

Le projet du "nouveau reporting" a pour objectif de mettre en place un système capable de réaliser le processus de reporting en tenant compte des critiques émises à l'encontre du système existant. Il tient également compte de l'évolution des technologies et de l'apparition de nouveaux logiciels depuis la mise en place du système de reporting sur mainframe. Dans ce chapitre, nous allons voir les principes de ce nouveau système ainsi qu'un de ses sous-projets qui nous concerne plus particulièrement à savoir la validation intelligente de données.

3.2 Les principes

L'idée de base du "nouveau reporting" consiste à rendre aux comptables un moyen d'agir sur le processus de reporting sans devoir faire appel aux informaticiens. En effet, on a vu que dans le système actuel, tout est automatisé avec les inconvénients que cela peut engendrer. Toutefois les idées appliquées pour le système existant sont reprises dans le "nouveau reporting".

Depuis la mise en place du système sur le mainframe, un nouveau logiciel offrant des fonctionnalités de reporting est apparu sur le marché. Le logiciel FiRE, pour

Financial **Re**porting et commercialisé par la société FRS, puise dans un ensemble de données mis à sa disposition celles nécessaires à la production de divers documents comptables destinés à l'entreprise, ses partenaires ou aux institutions de surveillance. Il se charge donc de réaliser la ventilation des données et il permet également à ses utilisateurs de paramétrer la présentation des documents selon les souhaits des destinataires de l'information. Pour cela, FiRE nécessite d'être alimenté en données comptables. Cette alimentation doit se faire par l'intermédiaire de fichiers dont la structure est prédéfinie. L'acquisition de ce logiciel évite de devoir "inventer" les règles spécifiques à la production des états légaux alors qu'elles existent déjà. En effet, il n'est plus nécessaire de traduire les conventions édictées par la législation et les autorités de contrôle en règles de regroupement puisque FiRE les contient déjà. On comprendra que les règles de regroupement actuellement utilisées pour effectuer le reporting n'auront plus la même utilité. Toutefois, on doit maintenant créer les fichiers d'alimentation de ce logiciel, ce à quoi les règles serviront dorénavant. Elles n'ont donc plus la même fonction que dans le système existant mais elles restent tout aussi importantes.

Nous allons maintenant décrire le mécanisme de ce nouveau système tel qu'il est décrit sur la figure 3.1.

Le backup de la base de données, effectué chaque mois, est encore utilisé afin de créer le fichier `Detbilar`, que l'on a renommé `Rep_In`. Mais, afin de pouvoir répondre aux nouvelles exigences en matière d'informations, de nouvelles données viennent enrichir ce fichier (cet enrichissement est dénoté par l'objet "complément" sur la figure 3.1). Une fois les données recueillies, il faut encore produire les fichiers nécessaires à l'utilisation du logiciel FiRE. Des règles de regroupement sont donc définies à cet effet. Pour répondre à la critique du système existant, ces règles seront désormais gérées par les comptables eux-mêmes et non plus par les informaticiens, pouvant ainsi être affectés à d'autres tâches. Un outil de gestion des règles, inexistant dans le système actuel, devra être mis en place afin de faciliter ce travail de gestion. De plus, afin d'éviter les corrections manuelles fastidieuses et coûteuses en temps, un mécanisme

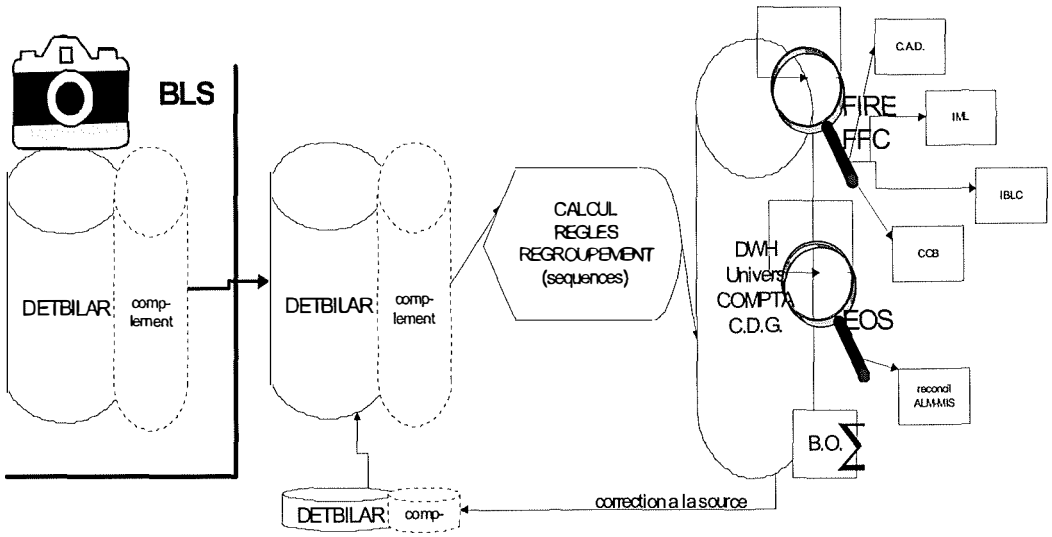


FIG. 3.1: Le nouveau reporting.

de correction des données sera mis en place, ce qui offrira la possibilité de ré-exécuter le programme afin de tenir compte des corrections de données ou de règles. Tout cela vise à produire les documents financiers nécessitant le moins de corrections manuelles a posteriori. Mais cela exige de pouvoir accéder aux données, de les visualiser ainsi que les différents documents sous forme électronique. Cette visualisation et cette correction à la source des données, c'est-à-dire directement dans le système d'informations, seront rendues possible grâce à l'introduction d'un datawarehouse. En effet, le datawarehouse ou entrepôt de données permet d'effectuer, en un point unique, un stockage intermédiaire des données issues des applications de production, dans lesquelles les utilisateurs finaux puisent avec des outils de restitution et d'analyse. Il permet à l'organisation d'exploiter rapidement et efficacement de grandes quantités d'informations, ce qui est rendu nécessaire par l'apparition et le développement de phénomènes économiques comme la mondialisation. Les entreprises évoluant dans un environnement difficile à appréhender, la prise de décision (stratégique ou politique) devient de plus en plus complexe à cause du nombre de paramètres à prendre en compte

et, en même temps, doit intervenir très rapidement pour ne pas laisser le temps aux concurrents de prendre de l'avance. L'introduction d'un datawarehouse permet donc aux utilisateurs de disposer et de visualiser toutes les informations à l'aide d'un outil, par exemple Business Object [1], et de réaliser eux-mêmes le formatage des documents. Les employés peuvent désormais accéder à l'information, ce qui permet à l'entreprise d'être plus réactive mais pose aussi de nouveaux problèmes comme les problèmes de confidentialité de l'information, de compétences pour l'analyse, ...
Les données contenues dans le datawarehouse proviendront de la base de données de la banque. Cette alimentation du datawarehouse sera également effectuée à l'aide des règles de regroupement. Nous voyons ainsi que, dans ce nouveau système, les règles de regroupement peuvent avoir de nombreuses fonctions.

3.3 L'architecture globale et ses composants

La figure 3.1 offre une vue générale du nouveau système. Nous allons maintenant nous intéresser à une partie de ce nouveau système, dont l'architecture est présentée à la figure 3.2 et dans laquelle l'outil de gestion des règles est représenté.

A partir de la base de données de la banque, des informations comptables seront extraites permettant ainsi la création du fichier `Rep_In`. Pour pouvoir traiter ces informations afin de les rendre disponibles à FiRE, de pouvoir les introduire dans le datawarehouse, ..., nous devons définir des règles de regroupement. L'outil de gestion de règles, que nous appellerons par la suite "éditeur de règles", le permettra. Les traitements appliqués sur les données du fichier `Rep_In` à l'aide des règles de regroupement sont paramétrés par divers fichiers qui sont au nombre de cinq et représentés sur la figure 3.2 :

[1] outil utilisé par la banque Dexia-BIL et proposé par la société Business Object

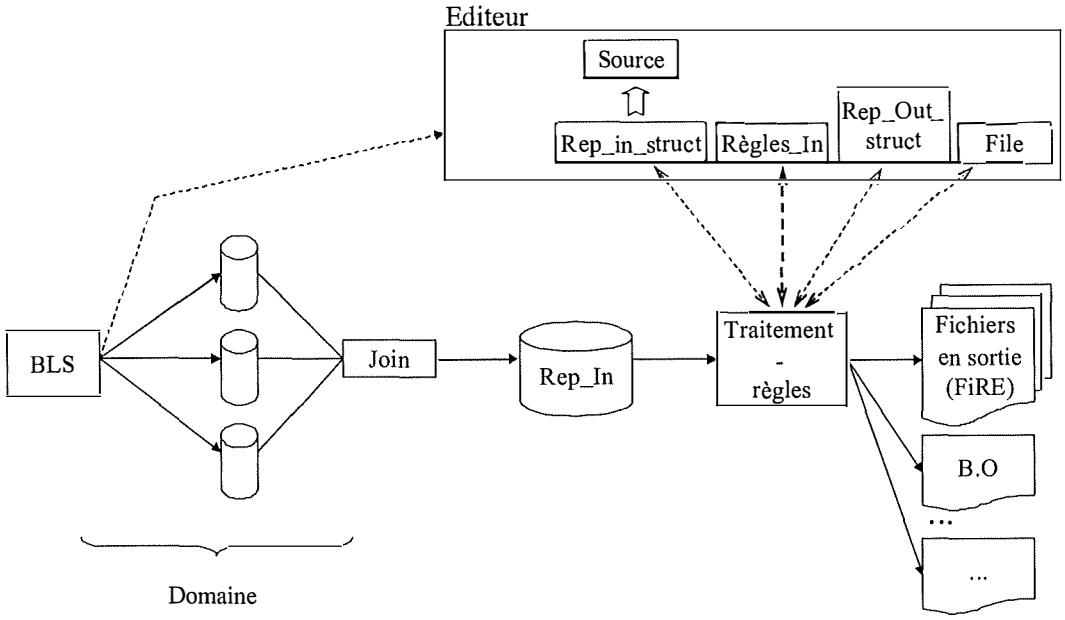


FIG. 3.2: L'architecture globale du système.

1. `Regle_In` qui contient les règles de regroupement définies par les comptables ;

2. `Rep_In_Struct` qui contient la structure du fichier `Rep_In` et dont un extrait est proposé à la figure 3.3. Cette structure indique le nom des informations devant y apparaître, leurs localisations dans le fichier (en indiquant la position du premier caractère de l'information dans le fichier), leurs tailles ainsi qu'un nom abrégé facultatif. Par exemple, l'information `Identifiant_Detail_Bilantaire` sera localisée au caractère 0 du fichier `Rep_In` et a une taille de 16 caractères. Cela signifie donc que la seconde information du fichier, `Code_Institut_Rapport`, sera localisée au $16^{ème}$ caractère du fichier et ainsi de suite comme on peut le voir à la figure 3.4 ;

	Nom	Debut	Taille	
0	Identifiant_Detail_Bilantaire	0	16	id_detb
1	Code_Institut_Rapport	16	2	code_inst
2	Type_Rapport	18	2	type_rap
3	Numero_Rapport	20	16	num_rap
4	Type_Ordre	36	2	type_ord
5	Numero_Ordre_Staging	38	9	

FIG. 3.3: Extrait de la structure de `Rep_In`.

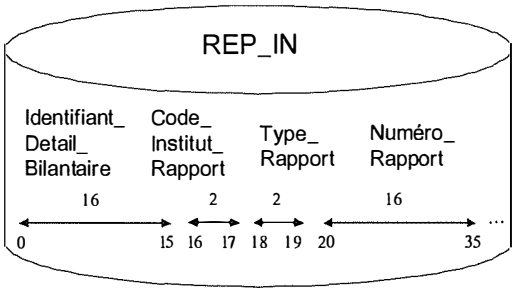


FIG. 3.4: Le fichier `Rep_In`.

3. `Source` indique la source des informations apparaissant dans le fichier `Rep_In`. En effet, nous ne sommes pas en mesure de localiser ces informations dans la base de données à l'aide des renseignements fournis par le fichier `Rep_In_Struct`. Le fichier `Source` permet cette localisation en indiquant, entre autres, pour chaque information la table qui la contient dans la base de données. Un extrait est proposé à la figure 3.5. ;

Table	Colonne	Type	Taille
DETAIL_BILANTAIRE	Identifiant_Detail_Bilantaire	varchar	16
DETAIL_BILANTAIRE	Code_Institut_Rapport	varchar	2
DETAIL_BILANTAIRE	Type_Rapport	smallint	2
DETAIL_BILANTAIRE	Numero_Rapport	varchar	16
DETAIL_BILANTAIRE	Type_Ordre	varchar	2
DETAIL_BILANTAIRE	Numero_Ordre_Staging	varchar	9

FIG. 3.5: Extraits des informations du fichier `Source`.

4. `Rep_Out_Struct` définit des séquences d'informations. Ces séquences sont constituées à partir des regroupements réalisés grâce aux règles conservées dans `Regle_In`. Par exemple, nous pouvons voir sur la figure 3.6 que la séquence `Sequence_1` est définie à partir des regroupements, nommé champs sur cette même figure 3.6, `Id1`, `Id2`, `Id3`, ..., `C013` et ce dans l'ordre indiqué par les nombres dans la colonne correspondante. Donc, la séquence `Sequence_1` est composée, dans cet ordre, par les champs `Id1`, `Id2`, `Id3`, `Id4`, `Id5`, `Id6`, `C013`, `Col1`, `Col2`, `Col3`, `Col4`, `Col5`, et enfin `Col6`.

Out	Champs	Typ_for	Longueur	Precision	Sequence_1	Sequence_4	Type_1	Type_4	Dont_1
1	Id1	A	16		0	0			
2	Id2	A	16		1	1			
3	Id3	A	4		2	2			
4	Id4	N	6		3	3			
5	Id5	N	6		4	4			
6	Id6	A	4		5	5			
7	Col1	S	20	3	7				
8	Col1Bis	S	20	3		7			
9	Col2	S	20	3	8	8			
10	Col3	S	20	3	9	9			
11	Col4	S	20	3	10	10			
12	Col5	S	20	3	11	11			
13	Col6	S	20	3	12	12			
14	C006	N	3				0		0
15	C007	A	3				2	2	2
16	C008	N	1				3	3	3
17	C010	A	2				4	4	4
18	C011	N	4				5		5
19	C012	N	1				6		6
20	C013	N	8		6	6	7		

FIG. 3.6: Extraits des informations du fichier `Rep_Out_Struct`.

5. `File` reprend la liste des différents fichiers à créer, à savoir leurs noms, leurs types et les séquences d'informations qu'ils contiennent (colonne `logique` dans la figure 3.7) et définis dans le fichier `Rep_Out_Struct` ;

Logique	Physique	Nom	Type
Sequence_1	0	rep_out	FIX
Sequence_4	0	rep_out	FIX
Type_1	1	rep_out_fire1	FIX
Type_4	2	rep_out_fire4	FIX

FIG. 3.7: Extraits des informations du fichier `File`.

La figure 3.8 détaille le processus de création du fichier `Rep_In` tel qu'on l'a aperçu sur les figures 3.1 et 3.2 tandis que la figure 3.9 détaille la partie finale à savoir le processus de création des fichiers finaux grâce à l'application des règles de regroupement.

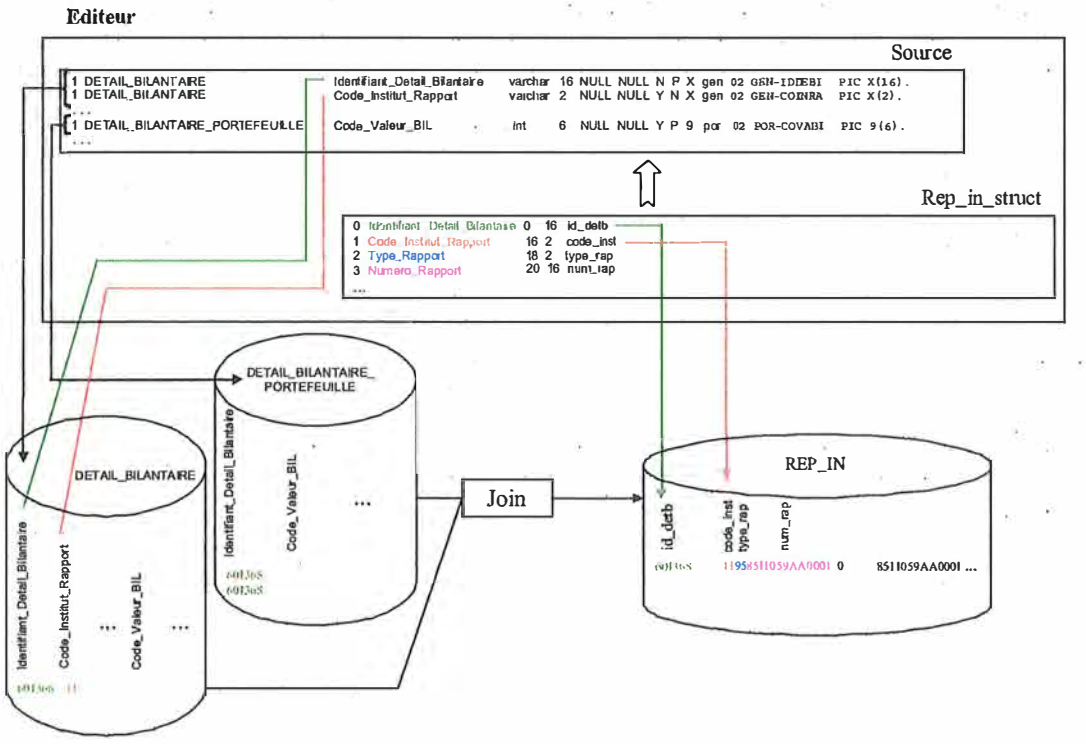


FIG. 3.8: Processus de création de `Rep_In`.

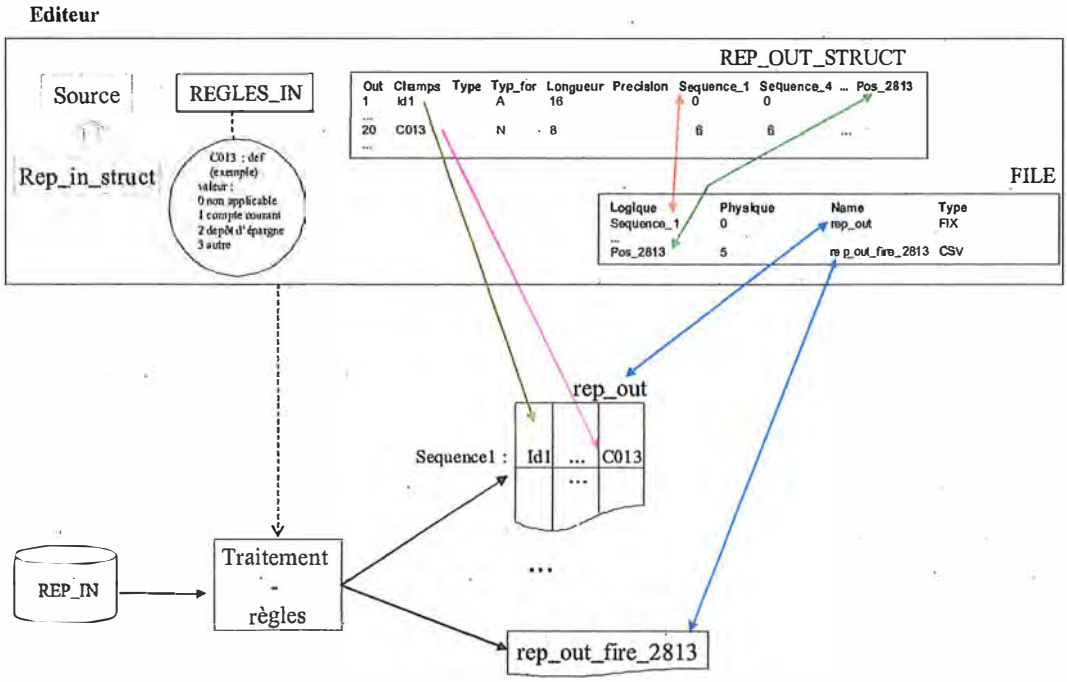


FIG. 3.9: Processus de création des divers fichiers finaux

3.3.1 L'éditeur de règles

Dans le système existant, nous avons vu qu'aucun outil de saisie n'est à disposition des informaticiens afin d'exprimer les règles de regroupement. Ceux-ci doivent donc jongler avec des codes identifiant les informations au lieu de pouvoir exprimer des règles à l'aide de libellés. Par exemple, un pays est identifié par un code. Les règles seront exprimées à l'aide de ce code plutôt qu'avec le libellé, à savoir le nom du pays. Comme le projet de "nouveau reporting" veut permettre aux comptables d'exprimer eux-mêmes les règles, l'introduction d'un outil de saisie convivial faciliterait l'expression des règles par tous. Il doit donc permettre à ses utilisateurs de gérer les règles de regroupement qui seront appliquées sur les informations récoltées dans `Rep_In`. De plus, l'augmentation du nombre de règles et l'importance des regroupements qu'elles définissent conduisent les utilisateurs à commettre des erreurs de saisies. Détecter ces erreurs lorsqu'elles ont été introduites dans le système demandera des efforts importants. C'est pourquoi on veut rendre cet outil de saisie "intelligent", c'est-à-dire permettant d'effectuer des vérifications quant à la validité des règles exprimées.

Notre attention durant le stage s'est focalisée principalement sur l'aspect de cette validation intelligente des règles. Le développement de l'éditeur de règles a donc été mis de côté. Toutefois, une réflexion sur les besoins de ses utilisateurs a été menée et a permis d'élaborer une structure pour celui-ci, détaillée à la figure 3.10.

L'objectif principal de cet éditeur est donc la gestion de règles. Celle-ci doit permettre :

- l'ajout d'une nouvelle règle ;
- la suppression d'une règle existante ;
- la modification d'une règle existante.

Toutes ces règles seront stockées en un endroit unique. Il a été décidé de les stocker dans un fichier de format XML. Ce format permet en effet une adaptation facile en cas d'évolution de la structure des règles ou des besoins.

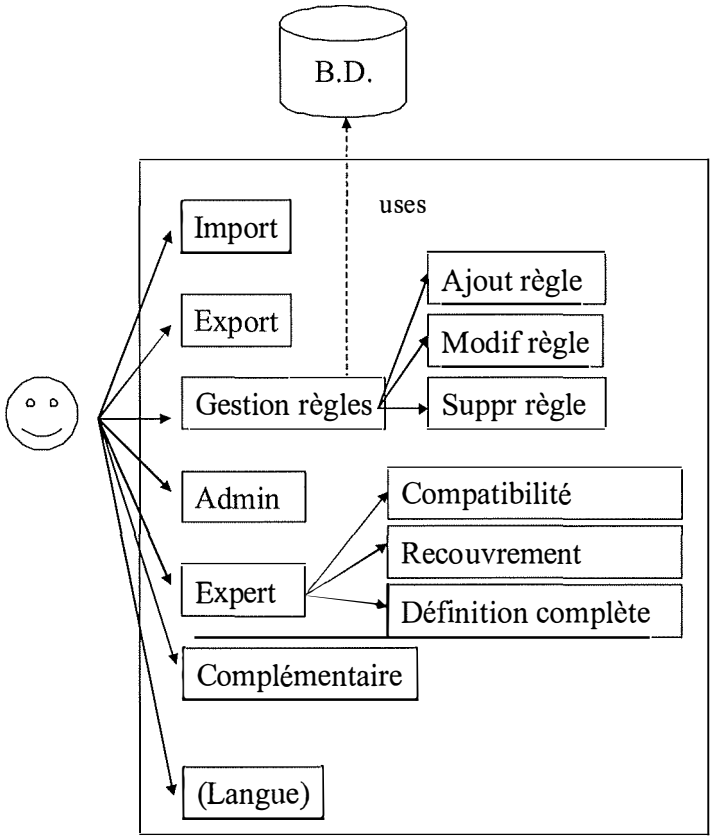


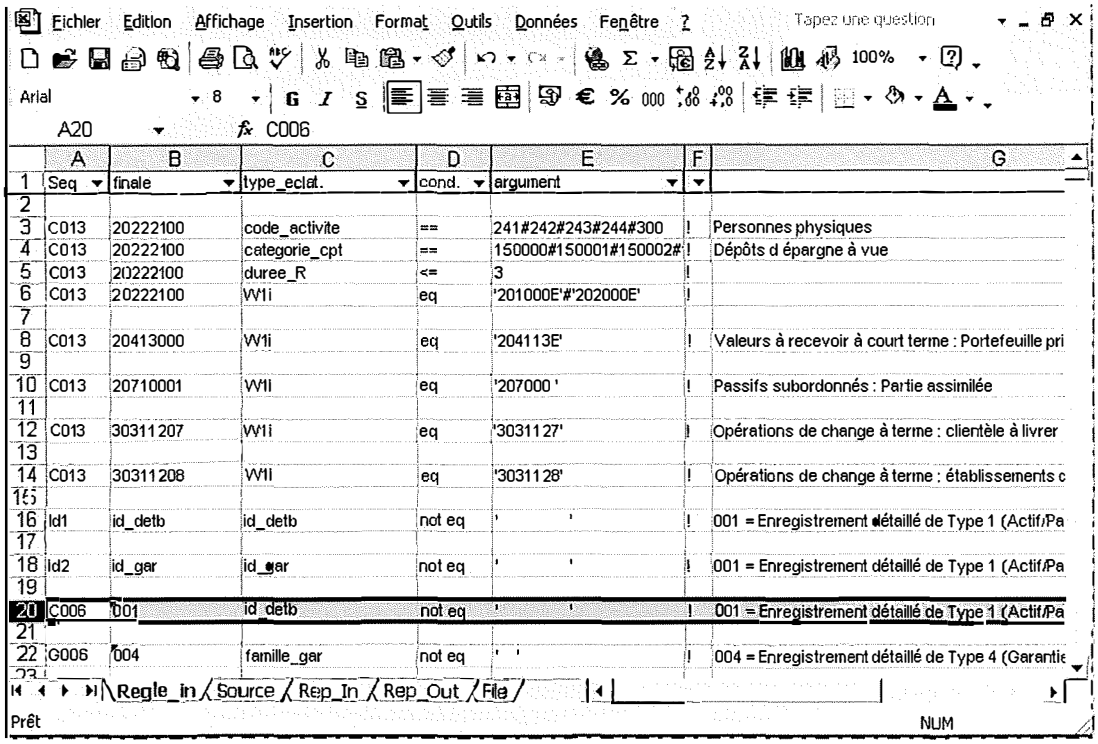
FIG. 3.10: Structure de l'éditeur de règles.

Il doit aussi être possible d'importer des règles ou de les exporter sous un format de fichier différent. On pourrait aussi imaginer de garder une copie de sécurité du fichier des règles lorsqu'un utilisateur effectue des modifications. Cette fonction permettrait aussi de garder un historique des modifications effectuées afin de pouvoir éventuellement revenir en arrière et annuler une modification.

Toutefois, même si l'éditeur de règles a pour objectif de permettre aux utilisateurs d'entreprendre par eux-mêmes des actions afin d'influencer le processus de reporting, ce qu'ils ne pouvaient faire auparavant, certains paramètres ne pourront leur être accessibles. Par exemple, les structures des fichiers ne pourront être modifiée que par

les développeurs. C'est pourquoi, nous avons prévu une partie d'administration. Par contre les utilisateurs pourraient paramétrer certains aspects de l'éditeur de règles afin d'en améliorer la convivialité. Par exemple, étant donné le caractère multilingue de la banque, on peut penser à offrir la possibilité de paramétrer la langue d'utilisation de l'éditeur afin d'améliorer la prise en main de celui-ci. La convivialité suppose aussi de pouvoir travailler sur des informations dont la dénomination est intuitive, par exemple en utilisant les libellés des informations. La base de données permettra cela en récupérant des renseignements sur ces informations.

Un premier prototype d'éditeur de règles a été réalisé à l'aide de Microsoft Excel. La figure 3.11 nous donne un aperçu de cet éditeur.



	Seq	finale	type_eclat.	cond.	argument		
3	C013	20222100	code_activite	==	241#242#243#244#300	!	Personnes physiques
4	C013	20222100	categorie_cpt	==	150000#150001#150002#	!	Dépôts d épargne à vue
5	C013	20222100	duree_R	<=	3	!	
6	C013	20222100	W1i	eq	'201000E'#'202000E'	!	
8	C013	20413000	W1i	eq	'204113E'	!	Valeurs à recevoir à court terme : Portefeuille pri
10	C013	20710001	W1i	eq	'207000 '	!	Passifs subordonnés : Partie assimilée
12	C013	30311207	W1i	eq	'3031127'	!	Opérations de change à terme : clientèle à livrer
14	C013	30311208	W1i	eq	'3031128'	!	Opérations de change à terme : établissements c
16	Id1	id_detb	id_detb	not eq	' '	!	001 = Enregistrement détaillé de Type 1 (Actif/Pa
18	Id2	id_gar	id_gar	not eq	' '	!	001 = Enregistrement détaillé de Type 1 (Actif/Pa
20	C006	001	id_detb	not eq	' '	!	001 = Enregistrement détaillé de Type 1 (Actif/Pa
22	G006	004	famille_gar	not eq	' '	!	004 = Enregistrement détaillé de Type 4 (Garantie

FIG. 3.11: L'éditeur de règles.

Ce prototype offre des fonctionnalités de base comme la gestion des règles mais

n'est pas encore suffisamment convivial. Par exemple, on peut voir sur l'aperçu que les pays sont exprimés par des codes et non par leurs libellés.

L'éditeur de règle doit offrir un support intelligent à ses utilisateurs. C'est ce que nous dénommons "Expert".

3.3.2 La validation intelligente des règles

La partie "Expert" de l'éditeur de règle doit donc permettre une validation intelligente des règles de regroupement définies par l'utilisateur. La validation consiste en diverses vérifications effectuées sur les règles définies. Elles doivent permettre à l'utilisateur de détecter d'éventuelles erreurs d'expression des règles et diminuer le risque d'incohérence dans les documents financiers à produire. Pour rappel, la validation permettrait de supprimer une partie des erreurs de formulation des règles de regroupement, réduisant ainsi l'ampleur du travail de recherche à effectuer lorsqu'une incohérence est détectée dans les documents produits.

La suite de ce travail ne traitera que de cette partie.

Chapitre 4

Spécifications du problème de validation intelligente de règles comptables

4.1 Introduction

Maintenant que nous avons une connaissance globale du système qu'on souhaiterait mettre en place, nous allons spécifier le problème qui nous concerne, à savoir la validation intelligente des règles exprimées par les comptables. Dans ce chapitre, nous verrons d'abord une description des règles de regroupement. Pour ce faire, nous nous inspirerons de la grammaire *Backus Naur Form* (BNF).
Différents besoins concernant la validation intelligente des règles ont pu être identifiés en collaboration avec les utilisateurs de l'outil de gestion de règles. Nous les définirons et les formaliserons mathématiquement.

4.2 Description des règles de regroupement

4.2.1 Quelques conventions de la grammaire *BNF*

Nous allons voir ici les différents symboles utilisés pour notre grammaire.
Les objets de la grammaire sont représentés de la manière suivante : <objet> La notation <objet>* signifie que nous sommes en présence de 0 ou plusieurs occurrences de l'objet indiqué. Un objet est défini à l'aide d'autres objets grâce à la notation <objet> ::= <objet$_1$> ...<objet$_n$> où le symbole "::=" établit la correspondance entre les deux parties.

4.2.2 Les règles de regroupement

Jusqu'ici nous avons parlé des règles de regroupement sans les aborder en détail. Pour pouvoir comprendre le problème de la validation, il nous faut d'abord expliquer en quoi consistent ces règles. Ces règles de regroupement doivent s'appliquer sur les données du fichier `Rep_In`, contenant des informations de la banque. Elles permettent de classer les informations selon les exigences définies par la législation. Chaque classification est composée de catégories parmi lesquelles les informations sont réparties selon des critères à observer. Ces critères peuvent être, par exemple, le pays d'origine des clients, l'activité du client, ...
Imaginons que l'on souhaite regrouper les informations des clients de la banque originaires de Belgique et exerçant l'activité d'établissement de crédit. Supposons que ces informations doivent être regroupées dans une classification nommée `C013`, sous la catégorie `1012000`. Nous pouvons exprimer cela de la manière suivante :

```
SI Pays='Belgique' et Activité='Etablissement de crédit'
ALORS C013=1012000
```

Cette définition peut être généralisée comme suit :

SI <condition> ALORS <classification>=<valeur> tel que
<condition> ::= **<variable> <operateur> <argument>**

Chaque variable possède un ensemble de valeurs qu'elle peut prendre, son domaine. Nous allons d'abord considérer qu'un seul argument peut-être spécifié dans une condition.

Mais afin de classer les informations correctement, plusieurs critères peuvent être nécessaire, comme dans l'exemple précédent, ce que ne nous permet pas cette définition. Il faut donc pouvoir exprimer une conjonction de critères ou conditions. Dès lors, nous définissons le concept de <sous-regle>. Une sous-règle représente une conjonction de critères et peut être représentée comme suit :

<sous-regle> ::= **<condition>** (**et** **<condition>**)*

et une règle de regroupement devient :

SI <sous-regle> ALORS <classification>=<valeur>

Toutefois, cette définition n'est pas encore complète. En effet, il est possible que l'on veuille regrouper des informations différentes et vérifiant des critères distincts sous la même catégorie d'une classification. Nous avons besoin, pour cela, de pouvoir exprimer des disjonctions de sous-règles. Nous définissons alors le concept de <regle> telle que

<regle> ::= **<sous-regle>** (**ou** **<sous-regle>**)*

et une règle de regroupement devient alors :

SI (<regle>) ALORS <classification>=<valeur>

Cette définition des règles de regroupement présente encore un désavantage. En effet, nous avons restreint l'écriture d'une condition à un seul argument. Mais les exigences légales imposent parfois des regroupements tels qu'une variable doit prendre une valeur parmi un ensemble de valeurs possibles. Par exemple il est possible de devoir regrouper les informations sous une classification et une catégorie déterminées si le pays d'origine du compte géré se retrouve parmi ceux de la liste indiquée, par exemple [Belgique, Luxembourg]. Si nous ne pouvons stipuler qu'un seul argument dans une condition, nous devrions écrire autant de conditions qu'il existe d'arguments à vérifier.

En reprenant le dernier exemple donné, nous devrions alors définir la règle de regroupement comme suit :
SI Pays='Belgique' ou Pays='Luxembourg' ALORS <classification>=<valeur>.

De cette manière, l'écriture des règles deviendra vite fastidieuse lorsque le nombre d'arguments à énumérer sera élevé. La solution est de pouvoir écrire une seule condition dans laquelle tous les arguments seront réunis sous la forme d'une liste. Nous levons pour cela la restriction posée précédemment et l'exemple devient alors

SI Pays='Belgique' OU 'Luxembourg' ALORS <classification>=<valeur>.

Par facilité, on introduit une nouvelle convention de notation "#" représentant le "OU" et nous permettant d'écrire :
SI Pays='Belgique' # 'Luxembourg' ALORS <classification>=<valeur>.
Nous devons donc modifier la définition de la condition comme suit :

<condition> ::= **<variable>** **<operateur>** **<argument>**(#**<argument>**)*
Le symbole "#" a été choisi en sachant qu'il était impossible que celui-ci apparaisse dans un argument.

Nous allons illustrer ces propos par un extrait des règles de regroupement définies dans le cadre du projet du "nouveau reporting" et proposé à la figure 4.1. Cet exemple nous met en présence d'une classification (`C013`) composée de deux catégories (`10613000` et `10621000`) définies chacune par une règle. La règle concernant la catégorie `10621000` est quant à elle composée de deux sous-règles.

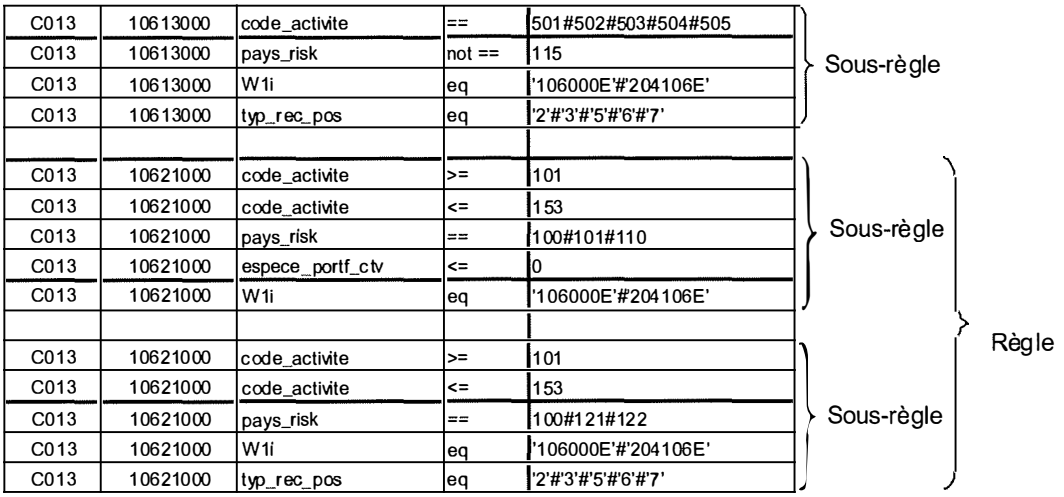


FIG. 4.1: Exemple de règles de regroupement.

Afin de poser clairement le problème et éviter des ambiguïtés, nous définirons ici quelques concepts importants pour la suite :

1. Une **expression booléenne (EB)** est de la forme <*variable*> <*opérateur*> <*arguments*> où
   - un argument est la valeur que l'on attribue à la variable. Cette valeur peut être soit un nombre, soit un string.
   - un opérateur est repris dans la liste non exhaustive suivante
     $\{=, <, \leq, >, \geq, \neq,$ *eq*, *not eq*$\}$ où `eq` et `not eq` permettent d'effectuer des comparaisons avec des arguments de type string

   Un expression booléenne n'est rien d'autre que ce que nous avons appelé auparavant une condition.
2. Une **sous-règle (SR)** est composée d'une expression booléenne ou de la conjonction de plusieurs expressions booléennes.
3. Une **règle (R)** est composée d'une sous-règle ou de la disjonction de plusieurs sous-règles.
4. Une **classification (CL)** est définie par une règle ou par la disjonction de

plusieurs règles.

5. Le domaine d'une variable, noté *dom(variable)*, représente l'ensemble des arguments que l'on peut affecter à une variable.

Nous pouvons formaliser ces concepts de la manière suivante.
Soit les indices suivant :
i $\in \{1..n\}$ représentant le numéro de la classification CL ;
j $\in \{1..m\}$ représentant le numéro de la règle R dans la classification CL_i ;
k $\in \{1..p\}$ représentant le numéro de la sous-règle SR dans la règle R_{ij} ;
l $\in \{1..q\}$ représentant le numéro de l'expression booléenne EB dans la sous-règle SR_{ijk}.

La figure 4.2 montre une représentation graphique de ces règles de regroupement.

$$CL_i \equiv \begin{cases} R_{i1} \\ \vdots \\ R_{ij} \equiv \begin{cases} SR_{ij1} \\ \vdots \\ SR_{ijk} \equiv \begin{cases} EB_{ijk1} \\ \vdots \\ EB_{ijkl} \\ \vdots \\ EB_{ijkq} \end{cases} \\ \vdots \\ SR_{ijp} \end{cases} \\ \vdots \\ R_{im} \end{cases}$$

FIG. 4.2: Représentation des règles de regroupement.

Nous définirons ensuite l'ensemble des solutions obtenues par l'évaluation d'une expression booléenne, d'une sous-règle, d'une règle, voire d'une classification. Pour

cela, nous avons besoin de définir l'ensemble des variables ayant au moins une occurrence dans une expression booléenne, dans une sous-règle, dans une règle ou dans une classification.
$vars(EB_{ijkl})$ = ensemble des variables intervenant dans l'expression booléenne EB_{ijkl} }. Par définition de l'expression booléenne, celle-ci n'est définie que sur une seule variable et donc l'ensemble $vars(EB_{ijkl})$ est un singleton.

$vars(SR_{ijk}) = \bigcup_{l=1}^{q} vars(EB_{ijkl})$
= ensemble des variables intervenant dans les expressions booléennes EB_{ijkl} formant la sous-règle SR_{ijk}.

$vars(R_{ij}) = \bigcup_{k=1}^{p} vars(SR_{ijk})$
= ensemble des variables intervenant dans les sous-règles SR_{ijk} formant la règle R_{ij}.

$vars(CL_i) = \bigcup_{j=1}^{m} vars(R_{ij})$
= ensemble des variables intervenant dans les règles R_{ij} formant la classification CL_i.

Nous définissons aussi la notation $EB_{ijkl}\{variable \setminus x\}$. Celle-ci représente l'évaluation de l'expression booléenne EB_{ijkl} telle que la variable `variable` $\in$ vars(EB_{ijkl}) est substituée par la valeur x $\in$ dom(`variable`). Le résultat de cette évaluation sera soit *vrai* soit *faux*.
Cette notation peut également être utilisée pour l'évaluation d'une sous-règle.

Nous pouvons maintenant définir les ensembles de solutions.

S(EB_{ijkl}), l'ensemble des solutions de l'expression booléenne EB_{ijkl}
= { x | variable $\in$ vars(EB_{ijkl}), x $\in$ dom(variable), $EB_{ijkl}\{variable \setminus x\}$ = true}
= l'ensemble des valeurs du domaine de la variable de l'expression booléenne telles

que celle-ci est vérifiée.

S(SR_{ijk}), l'ensemble des solutions de la sous-règle SR_{ijk}
$=\{ (x_1, \ldots, x_n) \mid \{X_1, \ldots, X_n\} = vars(SR_{ijk}),\ x_1 \in dom(X_1), \ldots, x_n \in dom(X_n),$
$SR_{ijk}\{X_1 \backslash x_1, \ldots, X_n \backslash x_n\} = \text{true}\}$
= l'ensemble des n-uplets de valeurs $(x_1, \ldots, x_n)$ tels que chaque élément x_i (1≤i≤n) correspond à une valeur pour la variable X_i et si on substitue les n variables par ces valeurs, la sous-règle sera vérifiée.

$S(R_{ij})$, l'ensemble des solutions de la règle R_{ij}
$= \bigcup_{k=1}^{p} S(SR_{ijk})$
= l'union des ensembles de solutions des sous-règles formant la règle R_{ij}.

S(CL_i), l'ensemble des solutions de la classification CL_i
$= \bigcup_{j=1}^{m} S(R_{ij})$
= l'union des ensembles de solutions des règles formant la classification CL_i

Par facilité, nous introduisons encore une nouvelle notation. Lorsqu'il est possible de substituer une variable par n'importe laquelle des valeurs de son domaine de sorte que la sous-règle puisse être vérifiée, nous noterons le nom de cette variable dans le n-uplet de solution. Cela permettra d'éviter d'écrire autant de solutions que de valeurs du domaine de cette variable.
Exemple :

`dom(X)={1,2,3,4},`
`dom(Y)={1,2,3,4,5},`
$X = 2 \wedge Y \geq 1.$

On voit que tous les couples de valeurs (2,1) (2,2) (2,3) (2,4) (2,5) sont des solutions de cette sous-règle. Nous pouvons abréger cette écriture en notant simplement (2,Y).

4.3 La compatibilité

4.3.1 Besoins

Lors de la définition des règles de regroupement, il peut arriver que l'utilisateur formule une règle pour laquelle il n'existera jamais de solution. Nous disons alors que cette règle est incompatible. Une telle règle n'a aucune utilité puisqu'elle sera toujours ignorée et résulte d'une erreur d'expression. Il est donc nécessaire de les détecter. Nous allons illustrer ce besoin par deux exemples.

Exemple 1 :

```
C013    10130000    code_activite     ==  140#141#142
C013    10130000    duree_R           <=  3
C013    10130000    duree_R           >   3
C013    10130000    W1I               eq  '103000E'#'104000E'
```

On voit très bien dans cet exemple qu'aucune solution ne pourra être trouvée par l'évaluation de cette sous-règle. En cause, les deux expressions booléennes impliquant la variable `duree_R` qui formulent des conditions contradictoires.

Exemple 2 :
Cet exemple introduit un autre type d'incompatibilité. Supposons que les domaines des variables et que la règle soient définis comme suit

```
dom(code_activite) = {140,141,142}
dom(duree_R) = {1,2,3}
dom(W1i) = {'103000E','104000E'}
```

```
C013    10130000    code_activite  ==  143
C013    10130000    duree_R        <=  3
C013    10130000    W1I            eq  '103000E'#'104000E'
```

On voit dès lors que pour la variable `code_activite`, aucune valeur du domaine ne permet de vérifier l'expression booléenne et donc la sous-règle.

Afin de simplifier la recherche d'éventuelles règles incompatibles, l'utilisateur doit avoir la possibilité de vérifier la compatibilité d'une sous-règle, d'une règle, d'une classification ou de toutes les classifications définies. Détecter une erreur, c'est bien mais pouvoir la corriger facilement, c'est encore mieux pour l'utilisateur. Pour ce faire, toute information permettant de localiser la source de l'incompatibilité ainsi que la cause est la bienvenue. Il faut donc identifier la ou les expressions booléennes à l'origine du problème ainsi que la sous-règle, la règle et la classification auxquelles elle(s) appartien(nen)t.

4.3.2 Définition

Pour les expressions booléennes, deux types de compatibilité peuvent être définis :

- **la compatibilité par rapport au domaine**
  une expression booléenne est compatible par rapport au domaine de sa variable si on peut trouver au moins une valeur la vérifiant dans le domaine de la variable.
  Exemple :
  Soit une variable X dont le domaine est défini par dom(X)={0,1,2,3,4,5}. L'expression booléenne $X < 5$ est compatible car on peut trouver des valeurs la satisfaisant dans le domaine de la variable X, à savoir les valeurs 0, 1, 2, 3 ou 4. L'expression booléenne $X > 5$ est incompatible car on ne peut trouver de valeurs la satisfaisant dans le domaine de la variable X.

- **la compatibilité entre expressions booléennes**
  Deux expressions booléennes appartenant à une même sous-règle, compatibles du point de vue de leurs domaines et possédant la même variable sont compatibles entre-elles si on peut trouver au moins une solution vérifiant la conjonction de ces expressions booléennes.
  Exemple :
  Soit la variable X de domaine dom(X)={0,1,2,3,4,5} et les expressions booléennes $X > 3$ et $X < 3$. La conjonction de ces deux expressions $X > 3 \wedge X < 3$ est incompatible étant donné qu'aucune solution ne peut être trouvée.

Dans la suite, nous dirons que des expressions booléennes sont compatibles si chacune est compatible par rapport à son domaine et si elles sont compatibles entre-elles.
La compatibilité des expressions booléennes étant définies, nous pouvons maintenant définir la compatibilité d'une sous-règle. Nous savons qu'une sous-règle est composée d'expressions booléennes, donc une sous-règle est compatible si les expressions booléennes la composant sont compatibles. Nous disons aussi qu'une règle est compatible si chacune de ses sous-règles est compatible. Une classification est dite compatible si chacune de ses règles est compatible.

4.3.3 Formalisation mathématique

Une sous-règle est compatible si et seulement si
$\forall EB_{ijka} \in SR_{ijk} : S(EB_{ijka}) \neq \emptyset$ avec $1 \leq a \leq l$
(compatibilité par rapport au domaine)
et
$\forall EB_{ijka}, EB_{ijkb} \in SR_{ijk}$ tel que $vars(EB_{ijka}) = vars(EB_{ijkb})$:
$S(EB_{ijka}) \cap S(EB_{ijkb}) \neq \emptyset$ avec $1 \leq a, b \leq l$ et $a \neq b$
(compatibilité entre expressions booléennes)

4.4 Le recouvrement

4.4.1 Besoin

Les informations comptables doivent être classées suivant les critères qu'elles observent dans une catégorie d'une classification. Dans l'idéal, une information ne peut être classée que sous une seule catégorie d'une seule classification. Si ce n'est pas le cas, cela signifie qu'il existe des règles de regroupement dont les critères se recouvrent, ce qui peut fausser les rapports financiers à produire. Cela ne se produira que lorsque des règles de regroupement partagent au moins une variable commune. Nous allons illustrer ce besoin à l'aide d'un exemple.
Soit la classification `C013` composée de deux catégories exprimées à l'aide des deux règles suivantes :

```
C013  10120000  code_activite ==  101
C013  10120000  pays_risk    ==  100#112#115#116#122#359
C013  10120000  duree_R      <=  3
C013  10120000  W1i          eq  '103000E'

C013  10331100  code_activite  ==  101
C013  10331100  duree_R        <=  3
C013  10331100  pays_risk      ==  116#117
C013  10331100  W1i            eq  '103000E'#'104000E'
```

Dans cet exemple, les deux règles de la classification *C013* se recouvrent pour les valeurs (`code_activite, pays_risk, W1i, duree_R`) = {101, 116, 103000E, `duree_R`}. Cela signifie qu'une information comptable vérifiant cette valeur peut être rangée dans la catégorie *10120000* ou dans la catégorie*10331100*. Donc, en cas de recouvrement, les informations pourraient ne pas être correctement ventilées dans les documents financiers. En effet, tout dépendra de l'ordre de sélection des règles de regroupement, ce qui risque de rendre encore plus difficile la recherche manuelle d'une erreur. Si les

règles sont traitées dans un ordre séquentiel, voie dans laquelle s'est orientée Dexia-BIL, les informations seront classées dans la catégorie exprimée par la dernière règle prise en compte. D'où l'importance de pouvoir détecter ces recouvrements dès la saisie des règles.

Deux types de recouvrement ont été identifiés :

- le recouvrement entre deux sous-règles appartenant à une même règle ;
- le recouvrement entre deux sous-règles appartenant à deux règles différentes. Ces deux règles doivent appartenir à la même classification.

Le premier cas ne génèrera pas d'erreur puisque les deux sous-règles appartiennent à la même règle et donc concernent une même catégorie mais l'écriture de la règle pourrait être simplifiée. Par exemple, les deux sous-règles suivantes

```
C013  10120000  code_activite ==  101
C013  10120000  duree_R      <=  3
C013  10120000  pays_risk    ==  115#116#122
C013  10120000  W1i          eq  '103000E'

C013  10120000  code_activite  ==  101
C013  10120000  duree_R       <=  3
C013  10120000  pays_risk     ==  116#117
C013  10120000  W1i           eq  '103000E'
```

peuvent être simplifiées comme suit :

```
C013  10120000  code_activite  ==  101
C013  10120000  duree_R       <=  3
C013  10120000  pays_risk     == 115#116#122#117
C013  10120000  W1i           eq '103000E'
```

Toutefois, cette simplification n'est pas toujours possible. En effet, il faut que les expressions booléennes qui ne participent pas au recouvrement soient exactement

identiques. Dans le cas contraire, la simplification ne sera pas possible. Par exemple, les règles

```
C013  10120000  code_activite ==  101
C013  10120000  duree_R      <=  3
C013  10120000  pays_risk    ==  115#116#122
C013  10120000  W1i          eq  '103000E'

C013  10120000  code_activite  ==  101
C013  10120000  duree_R       <=  3
C013  10120000  pays_risk     ==  116#117
C013  10120000  W1i           eq  '103000E'#'104000E'
```

ne peuvent être simplifiées car les expressions booléennes impliquant la variable `W1i` ne sont pas identiques. Si nous essayons de simplifier de la même manière, la nouvelle règle

```
C013  10120000  code_activite  ==  101
C013  10120000  duree_R       <=  3
C013  10120000  pays_risk     ==  115#116#117#122
C013  10120000  W1i           eq  '103000E'#'104000E'
```

possédera des solutions qui n'appartiennent pas à l'union des ensembles de solutions de ses deux règles constituantes. Par exemple, les solutions
(`code_activite, pays_risk, W1i, duree_R`) = { 101, 115, 104000E, `duree_R` },
{ 101, 122, 104000E, `duree_R` } sont des solutions qui n'étaient vérifiées par aucunes des deux règles.

Le deuxième cas de recouvrement identifié est le plus important car il peut être source d'erreur lors de la génération des états financiers en classant les informations sous une catégorie non appropriée.

Le gestionnaire doit pouvoir tester l'existence d'un recouvrement entre deux règles, dans une classification ou dans toutes les classifications définies. Dans le cas d'un test sur une classification, il faut voir la classification comme un ensemble de règles à tester entre-elles, deux à deux.
Pour éliminer un problème de recouvrement, l'utilisateur a besoin de renseignements concernant la localisation de ce problème, c'est-à-dire la classification et la ou les règle(s) auxquelles les sous-règles appartiennent, ainsi que sa cause. Connaître la cause du problème consiste à identifier les sous-règles pour lesquelles il existe une solution commune ainsi que cette solution.

4.4.2 Définition

Il existe un recouvrement entre deux sous-règles SR_{ian} et SR_{ibm}, avec $n \neq m$, si celles-ci possèdent au moins une solution commune. Cette solution appartient donc à l'intersection de leurs ensembles de solutions.

Si ces deux sous-règles appartiennent à la même règle, c'est-à-dire si $a = b$, une simplification est peut-être envisageable. Par contre, si elles appartiennent à deux règles différentes d'une même classification, le problème en découlant doit être corrigé.
A partir de là, nous pouvons définir le recouvrement entre deux règles d'une même classification. Il existe un recouvrement entre deux règles d'une même classification si l'intersection de leurs ensembles de solution est non vide. S'il existe un recouvrement entre deux règles, cela signifie aussi qu'il existe deux sous-règles, une dans chaque règle, qui se recouvrent. On doit donc tester, deux à deux, toutes les sous-règles de ces deux règles afin de déterminer précisément la cause du recouvrement.

4.4.3 Formalisation mathématique

Il y a recouvrement entre deux sous-règles SR_{ian} et SR_{ibm} si
$S(SR_{ian}) \cap S(SR_{ibm}) \neq \varnothing$ avec $n \neq m$.
Il y a recouvrement entre deux règles d'une même classification si elles possèdent une solution commune, c'est-à-dire si $S(R_{ia}) \cap S(R_{ib}) \neq \varnothing$ avec $a \neq b$.

4.5 La définition complète

4.5.1 Besoin

Ce besoin permet de s'assurer que toutes les informations nécessaires à la création des documents financiers seront bien regroupées sous les classifications définies. Nous avons défini les expressions booléennes comme représentant des critères de regroupement des informations. Si une valeur du domaine d'une variable quelconque intervenant dans la définition des règles de regroupement n'est pas prise en compte par une expression booléenne, toute information du fichier `Rep_In` répondant à un critère vérifiant cet argument ne sera jamais traitée, faussant de cette manière les résultats du reporting.

Nous allons illustrer ces propos par un exemple simple. Supposons la classification `C013` définissant la catégorie *10120000* à l'aide d'une règle impliquant quatre variables `code_activite, pays_risk, duree_R, W1i` dont les domaines sont les suivants

- dom(code_activite) = {101} ;
- dom(pays_risk) = {100, 112, 115} ;
- dom(duree_R) = {0, 1, 2, 3} ;
- dom(W1i) = {'103000E'}.

La règle est la suivante :

```
C013  10120000  code_activite ==  101
C013  10120000  pays_risk    ==  100#112
C013  10120000  duree_R      <=  3
C013  10120000  W1i          eq  '103000E'
```

On peut voir que les n-uplets `(code_activite,pays_risk,duree_R,W1i)` = `(code_activite,115,duree_R,W1i)` ne seront jamais vérifiés par la règle. Donc, cette classification *C013* n'est pas complètement définie.

L'utilisateur peut demander de tester une classification donnée ou toutes les classifications définies. Dans ce dernier cas, chaque classification sera testée individuellement. Dans le cas où la définition d'une classification est incomplète, il est intéressant d'en connaître les raisons, c'est-à-dire les variables ainsi que les valeurs pour lesquelles elles ne seront jamais traitées par une règle de regroupement.

4.5.2 Définition

Nous pouvons définir la définition complète d'une classification par analogie aux fonctions partout définies. Nous allons d'abord rappeler brièvement la notion de fonction partout définie ainsi que celle de produit cartésien qui nous sera également utile pour formuler la définition.

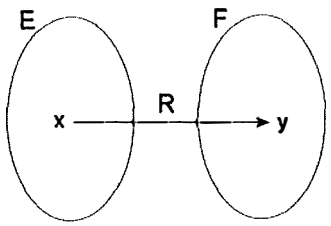


FIG. 4.3: Relation de E dans F.

Considérons deux ensembles non vides E et F. Si à certains éléments de E on peut associer par une règle mathématique précise R (non ambiguë) un élément y de F, on définit ainsi une relation de E vers F. On écrit :
$R : E \mapsto F$ et $x\ R\ y$
Soit l'ensemble D, l'ensemble des éléments de E qui ont au moins une image dans F par R. On l'appelle aussi ensemble de définition de R. Si D = E, c'est-à-dire si tous les éléments de E ont une image dans F par R, on dira que R est partout définie.

Le produit cartésien de n ensembles $A_1, A_2, \ldots, A_n$, noté $A_1 \times A_2 \times \ldots \times A_n$, est l'ensemble des n-uplets $(a_1, a_2, \ldots, a_n)$ tels que $a_i \in A_i$ avec i = 1, 2, ..., n et

$$|A_1 \times A_2 \times \ldots \times A_n| = |A1| * |A2| * \ldots * |An|.$$

Dans notre cas, l'ensemble E représente l'ensemble des solutions de la classification contrôlée, à savoir $S(CL_i)$ et l'ensemble F représente le produit cartésien des domaines des variables ayant au moins une occurrence dans cette classification.

Connaissant l'ensemble des variables intervenant dans la définition d'une classification ainsi que leurs domaines, celle-ci est dite complètement définie si l'ensemble des solutions de la classification est égale au produit cartésien des variables apparaissant dans cette classification, autrement dit $S(CL_i)$ = produit cartésien des domaines des variables tel que ces variables appartiennent à vars(CL_i). Donc toute valeur n'appartenant pas à l'ensembles des solutions de la classification mais bien au produit cartésien rend la classification incomplète.
Nous pouvons aussi utiliser la notion de complémentaire pour définir la définition complète. Pour rappel, si E et A sont deux ensembles, on appelle Complémentaire de A dans E l'ensemble formé des éléments de E qui ne sont pas dans A. On note $\mathrm{C}_E\mathrm{A}$ cet ensemble. Une représentation du complémentaire par diagramme est donnée à la figure 4.4.

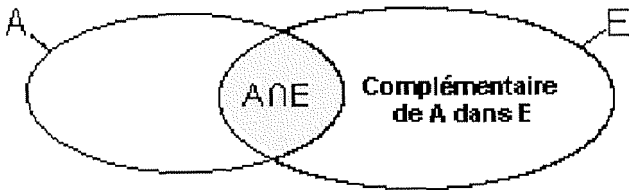


FIG. 4.4: Complémentaire d'un ensemble.

Nous pouvons donc définir la définition complète d'une classification à l'aide de la notion de complémentaire. En effet, si le complémentaire de l'ensembles des solutions d'une classification est vide par rapport au produit cartésien des domaines de ses variables, cette classification sera complètement définie.

4.5.3 Formalisation mathématique

Une classification CL_i est complètement définie si
$\forall(x_1, \ldots, x_n)$ *tels que* $\{X_1, \ldots, X_n\} = vars(CL_i)$, $x_1 \in dom(X_1), \ldots, x_n \in dom(X_n)$:
$(x_1, \ldots, x_n) \in S(CL_i)$, c'est-à-dire que tout élément du produit cartésien des domaines des variables appartient à l'ensemble des solutions de la classification. Autrement dit, le complémentaire de l'ensemble des solutions de la classification par rapport au produit cartésien des domaines des variables appartenant à vars(CL_i) (PCD) est vide, c'est-à-dire C_{PCD} S(CL_i) $\neq \emptyset$.

4.6 Le complémentaire

4.6.1 Besoin

Afin de définir plus facilement certaines règles, l'utilisateur aimerait disposer d'une fonction permettant de calculer le complémentaire d'une expression booléenne, d'une sous-règle ou d'une règle. Ce besoin ne concerne pas une validation des règles mais est juste un moyen d'exprimer plus rapidement et plus facilement certaines règles de regroupement. C'est donc une fonctionnalité offerte par l'éditeur de règles, n'appartenant pas à la partie `EXPERT` de celui-ci.

4.6.2 Définition

Expression booléenne

Calculer le complémentaire d'une expression booléenne consiste simplement à modifier son opérateur selon les correspondances suivantes :
$= \rightarrow \neq$
$\neq \rightarrow =$
$> \rightarrow \leq$
$< \rightarrow \geq$
$\geq \rightarrow <$

$\leq \rightarrow >$

$eq \rightarrow not\ eq$

$not\ eq \rightarrow eq$

Nous noterons le complémentaire d'une expression booléenne $\mathrm{C}(< expr.bool. >)$. Par exemple, le complémentaire de l'expression booléenne `duree_R` $\geq$ `3` sera `duree_R < 3`

Sous-règle

Une sous-règle est composée d'une expression booléenne ou de la conjonction de plusieurs expressions booléennes. Nous avons déjà vu comment calculer le complémentaire d'une expression booléenne. Nous savons aussi, par les règles de la logique, que le complémentaire d'une conjonction d'expressions booléennes revient à calculer la disjonction de ces expressions.
Sachant qu'une sous-règle est définie comme suit :
$< sous - regle > : := < expr.bool. >$ (et $< expr.bool. >)^*$,
le complémentaire de celle-ci sera :
$\mathrm{C}(< sous - regle >) : := \mathrm{C}(< expr.bool. >)$ (ou $\mathrm{C}(< expr.bool. >)^*$)

Nous voyons ici que nous aurons besoin de pouvoir distinguer les conjonctions d'expressions booléennes des disjonctions. Les disjonctions d'expressions booléennes seront exprimées en les délimitant par deux marqueurs "OU" afin de les différencier des conjonctions d'expressions booléennes.

Règle

Un règle étant composée d'une sous-règle ou de la disjonction de plusieurs sous-règles, son complémentaire sera égal au complémentaire de la sous-règle ou à la conjonction des complémentaires des sous-règles. En reprenant la définition d'une règle donnée au point 4.2 de ce chapitre
$< regle > : := < sous - regle >$ (ou $< sous - regle >)^*$,

nous représentons son complémentaire par
$\mathrm{C}(< regle >) : := \mathrm{C}(< sous - regle >)$ (et $\mathrm{C}(< sous - regle >)^*$)

Ensemble de règles

L'utilisateur désire également pouvoir calculer le complémentaire d'un ensemble de règles appartenant à une même classification. Celui lui permettrait, entre autres, de garantir que la classification contenant ces règles soit complètement définie. Cet ensemble représente la disjonction des règles et son complémentaire se calculera selon les mêmes principes que ceux vu pour le calcul du complémentaire d'une règle.

Nous allons illustrer tout cela en reprenant l'exemple défini au 4.5.1. L'utilisateur peut demander de calculer le complémentaire de la règle `10120000` afin de garantir la complétude de la classification `C013`. Il exprimerait donc C `10120000` et doit aussi spécifier la valeur de cette nouvelle catégorie exprimée, après calcul, par la nouvelle règle :

```
OU
C013  valeur  code_activite ≠  101
C013  valeur  pays_risk    ≠  100#112
C013  valeur  duree_R      >  3
C013  valeur  W1i          not eq  '103000E'
OU
```

4.7 Conclusion

Maintenant que nous avons défini les besoins de l'utilisateur, nous allons devoir trouver un moyen de résoudre ce problème efficacement. L'objectif du stage consistait à démontrer l'utilité de la programmation logique avec contraintes pour résoudre ce genre de problème. Le problème défini se prête bien à la programmation avec contraintes pour des raisons que nous serons en mesure de présenter lorsque nous aurons eu un aperçu de ce type de programmation.

Chapitre 5

La programmation logique avec contraintes

5.1 Introduction

Comme nous l'indique son nom, la programmation logique avec contraintes provient de la programmation logique. Dans ce chapitre, afin de mieux comprendre le style de la programmation logique avec contraintes, nous introduirons la programmation logique. Mais pour cela, nous découvrirons d'abord, brièvement, la théorie sur laquelle se fonde celle-ci. Une fois la théorie et la programmation logique abordées, nous pourrons aborder la programmation logique avec contraintes et ses différences. Nous illustrerons celles-ci à l'aide d'un exemple simpe. Pour clôturer ce chapitre, nous parlerons du langage ECL^iPS^e permettant de développer des applications à l'aide de la programmation avec contraintes.

5.2 La logique du premier ordre

Le lecteur pourra se reporter à [4] et à [5] pour de plus amples informations.

La logique propositionnelle

La logique propositionnelle ou calcul des propositions est une des théories les plus simples qui soient. Elle étudie des énoncés qui sont soit vrais, soit faux, à savoir des propositions. Par exemple, l'énoncé "il pleut" est une proposition car il est susceptible d'être soit vrai soit faux. Le vocabulaire de la logique propositionnelle est composée d'un ensemble de propositions désignées par une lettre minuscule (p,q, ...) et de cinq connecteurs :

$\neg$, représentant la négation ;

$\vee$, représentant la disjonction ;

$\wedge$, représentant la conjonction ;

$\Rightarrow$, représentant l'implication ;

$\Leftrightarrow$, représentant l'équivalence.

Ce vocabulaire donne lieu à des assemblages par juxtaposition de connecteurs et de propositions, appelés formules. Par exemple, $p \Rightarrow q$ est une formule.
Toutefois, la logique propositionnelle ne permet la formalisation que d'une petite partie de l'ensemble des raisonnements. En effet, la logique propositionnelle ne permet pas de rendre compte des raisonnements comme le suivant : "TOUT être humain est mortel, OR Socrate est un être humain, DONC Socrate est mortel" car il utilise les notions d'objet et de propriété des objets. Par exemple, dans "TOUT être humain est mortel", "mortel" est la propriété de l'objet "être humain" et dans "Socrate est mortel", "mortel" est la propriété de l'objet "Socrate".

La logique des prédicats ou logique du premier ordre

La logique des prédicats est une extension de la logique propositionnelle tenant compte de cette limitation. Un prédicat est une expression contenant 0, une ou plusieurs variables et qui est susceptible de devenir une proposition vraie ou fausse si on attribue à ces variables certaines valeurs. On voit ainsi qu'une proposition n'est rien d'autre qu'un prédicat sans argument. Son principal intérêt par rapport à la logique

propositionnelle est, donc, l'introduction des variables. Par exemple, supposons les trois propositions suivantes : r ("Il pleut"), u ("Jean prend son parapluie"), w ("Jean est mouillé"). Supposons de plus que nous ayons trois hypothèses que l'on considère vraies :

$r \Rightarrow u$ ("S'il pleut, alors Jean prend son parapluie")

$u \Rightarrow \neg w$ ("Si Jean prend son parapluie alors il ne sera pas mouillé")

$\neg r \Rightarrow \neg w$ ("S'il ne pleut pas, Jean n'est pas mouillé")

Ce qui est vrai pour Jean l'est aussi pour Marie, Alex, ... Nous pouvons imaginer la proposition u comme u_{Jean}, de même que w est la proposition w_{Jean}. Dans ce cas, nous avons les hypothèses

$r \Rightarrow u_{Jean}$

$u_{Jean} \Rightarrow \neg w_{Jean}$

$\neg r \Rightarrow \neg w_{Jean}$

Si on définit la proposition u_{Marie} signifiant Marie prend son parapluie, et w_{Marie} signifiant Marie est mouillée, alors nous avons le même ensemble d'hypothèses

$r \Rightarrow u_{Marie}$

$u_{Marie} \Rightarrow \neg w_{Marie}$

$\neg r \Rightarrow \neg w_{Marie}$

et ainsi de suite pour chaque personne que l'on connaît. L'introduction de la variable X permet d'écrire une seule proposition pour tout le monde dans laquelle X peut représenter Jean, Marie, Alex, ... :

$r \Rightarrow u_X$

$u_X \Rightarrow \neg w_X$

$\neg r \Rightarrow \neg w_X$

Le langage des prédicats est donc un langage plus riche que le langage de la logique propositionnelle. Il est constitué de :

- symboles de variables notés X, Y, Z,... ;
- symboles de constantes notés a, b, c,... ;

- symboles de fonctions notés f, g, h,... ;
- symboles de prédicats (ou relations) notés p, q, r,... ;
- symboles de connecteurs et des quantificateurs universel ($\forall$) et existentiel ($\exists$) ;
- parenthèses.

A chaque symbole de fonction et de prédicat est associé une arité, c'est-à-dire le nombre d'argument, qui est un entier positif ou nul. Si n est l'arité d'un prédicat p, nous pouvons aussi noter `p/n`. On peut considérer que les constantes sont des fonctions d'arité 0.

Les termes du langage sont définis de manière inductive. Un terme peut être :

- un symbole de constante ;
- un symbole de variable ;
- f(t_1,..., t_n) où f est un symbole de fonction n-aire et t_1, ..., t_n sont des termes.

Nous devons encore définir la notion d'atome, de littéral, de formule et de substitution afin de définir complètement le langage.

Un **atome** est un énoncé de la forme p(t_1,..., t_k) où p est un symbole de prédicat k-aire et t_1,..., t_k sont des termes.

Un **littéral** est soit un atome (littéral positif), soit un atome précédé du symbole de négation $\neg$ (littéral négatif).

Une **formule** dans la logique des prédicats peut être composée des différents éléments suivants :

- un littéral ;
- une quantification universelle ($\forall x$ .w) où x est une variable et w est une formule ;
- une quantification existentielle ($\exists x$ .w) où x est une variable et w est une formule ;
- des constructions $\neg(w)$, $(w1) \vee (w2)$, $(w1) \wedge (w2)$, $(w1) \Rightarrow (w2)$, $(w1) \Leftrightarrow (w2)$, où w, w1 et w2 sont des formules.

Dans une formule, on distingue les variables libres des variables liées. Une variable X dans une formule F est dite libre si elle ne se trouve dans aucune sous-formule de F

qui commence par $\forall$ ou $\exists$. Dans le cas contraire, elle est dite liée. Par exemple, dans la formule $\forall x\ p(x) \Rightarrow r(y,x)$ la variable x est liée tandis que la variable y est libre dans la formule.
L'opération de **substitution** consiste à remplacer certaines variables libres d'une formule w par des termes. Il s'agit d'une opération purement syntaxique. Soit w une formule où $x_1, \ldots, x_n$ sont des variables libres, σ la substitution $(t_1 \setminus x_1, \ldots, t_n \setminus x_n)$ et $w\sigma$ est équivalent à la formule w où toutes les occurrences des x_i sont substituées par t_i. Une substitution instancie la variable X si elle remplace X par un terme dans lequel n'apparaît aucune variable.

5.3 La programmation logique

La programmation logique est le fruit des recherches menées dans les années 1970 par R. Kowalski et A. Colmerauer. De ces travaux, le langage *Prolog*, acronyme de **PRO**grammation **LOG**ique et représentant de cette discipline, est né. La programmation logique touche aujourd'hui des champs d'applications variés comme la conception de systèmes experts, le traitement du langage naturel ou l'automatisme. Pour plus d'informations concernant le langage Prolog, le lecteur peut se reporter à [11] et à [10] concernant la programmation logique.

La programmation logique se base sur la logique du premier ordre, facilitant ainsi la modélisation des problèmes grâce aux relations et variables logiques. De ce fait, les langages de programmation logique sont de haut niveau et de nature déclarative. Son aspect déclaratif permet de résoudre simplement un problème en indiquant au système les données du problème à traiter, c'est-à-dire ce qu'il doit faire. En effet, les langages de programmation logiques utilisent des procédures de recherche prédéfinies permettant à l'utilisateur de se concentrer sur la modélisation. A titre de comparaison, les langages de programmation "classiques" tels que Pascal, Java ou C sont de nature impérative, ce qui signifie qu'il faut décrire le problème à résoudre selon un algorithme, c'est-à-dire indiquer au système comment faire pour résoudre le problème.

C'est pourquoi la programmation logique permet de résoudre des problèmes complexes en facilitant le formalisme ainsi que leurs résolutions. Elle autorise aussi un développement plus rapide que les autres langages grâce à sa sémantique proche des spécifications logiques du programme. La programmation logique permet également d'écrire des procédures non directionnelles, permettant de résoudre différents types de problèmes. Par exemple, considérons le prédicat append(L1,L2,R) permettant de concaténer deux listes L1 et L2 et donnant comme résultat R.
append([a,b,c], [d,e], R) $\rightarrow$ R=[a,b,c,d,e] mais il est aussi possible de résoudre d'autres problèmes comme
append([a],L2,[a,b,c]) $\rightarrow$ L2=[b,c] ou
append(L1,[c],[a,b,c]) $\rightarrow$ L1=[a,b].

Toutefois, quelques inconvénients sont à relever comme la lenteur d'exécution des programmes logiques et leur pauvreté en représentation de données. Les seules structures de données existantes en programmation logique sont les termes tels qu'on les a définis à la section 5.2.

Pour réaliser un programme logique permettant de résoudre un problème donné, nous devons modéliser nos connaissances du problème. La modélisation consiste à décrire les objets du problème ainsi que leurs relations, les relations décrivant les propriétés des objets et leurs interactions. Une variable permet de représenter un objet pouvant prendre des valeurs dans un certain domaine, comme une inconnue dans une équation. Les relations sont définies soit implicitement en terme de règles soit explicitement comme un ensemble de faits.

Les **règles** définissent une relation de manière générale, à partir d'autres relations. En programmation logique, une relation est représentée par un prédicat. Il est de la forme $p(X_1, \ldots, X_n)$ où p est le nom du prédicat et $X_1, \ldots, X_n$ sont les arguments. Chaque règle est composée de deux parties à savoir une tête et un corps et sa syntaxe est la suivante :

`tête :- corps.`

La tête est constituée d'un atome nommant la relation définie. Il joue le même rôle, syntaxique, que les définitions et les appels de procédures dans les langages classiques. Le corps est composé de relations aussi nommées sous-buts et représente la définition réelle de la relation. Le symbole `:-` est appelé le cou car il relie la tête et le corps.

Par exemple, supposons que nous voulons écrire un petit programme capable de déterminer l'espèce d'un animal d'après certaines de ses caractéristiques. A partir de la définition suivante :

"La girafe est un animal à longues pattes ayant un long cou et présentant des taches sombres ainsi que des rayures noires.",

nous pouvons traduire en logique du premier ordre

$\forall X$: `espece(X, girafe)` $\leftrightarrow$ `longues_pattes(X)` $\wedge$ `long_cou(X)`

$\wedge$ `taches_sombres(X)` $\wedge$ `rayures_noires(X)`.

Nous pouvons exprimer cette règle comme suit : "pour tout X tel que les relations `taches_sombres(X)`, `longues_pattes(X)`, `long_cou(X)`, et `rayures_noires(X)` existent, la relation `espece(X, girafe)` existe, c'est-à-dire que X est de l'espèce girafe". Sachant que la conjonction peut être représentée par une virgule en programmation logique, nous pouvons ensuite traduire :

```
espece(X,girafe) :- taches_sombres(X), longues_pattes(X), long_cou(X),
                    rayures_noires(X).
```

Si nous voulons introduire une règle permettant de décrire l'espèce guepard, nous définirons une règle de tête `espece(X, guepard)`. Un ensemble de règles consécutives ayant le même nom est appelé une procédure et se lit comme une disjonction. Par exemple,

```
espece(X, girafe) :- <sous-buts>
espece(X, guepard) :- <sous-buts>
```

peut être lue comme : "l'objet X peut être soit une girafe soit un guepard à condition de vérifier les sous-buts correspondants".

Un **fait** est un cas particulier d'une règle et il exprime soit une propriété universelle soit une caractéristique des données du problème à résoudre. Il établit donc explicitement une relation. Par exemple, l'énoncé "La Terre tourne" est un fait car c'est une propriété universelle. Sa syntaxe est la suivante :

`p.`

où p est un atome. En reprenant l'exemple donné ci-dessus et sachant que Nestor est une girafe, nous aurons alors les faits suivants :

```
longues_pattes(nestor).
long_cou(nestor).
taches_sombres(nestor).
rayures_noires(nestor).
```

On remarquera que Nestor est une constante or en programmation logique, les constantes doivent commencer par une minuscule d'où l'écriture "nestor".

L'ensemble des faits et des règles constitue le programme aussi appelé base de connaissance. Ce programme est utilisé pour répondre à des requêtes au sujet du monde décrit. Un programme logique peut être assimilé à une théorie dont l'exécution consiste à trouver une preuve que la relation indiquée dans la requête existe en la déduisant des faits et règles du programme. Les déductions sont effectuées à l'aide de l'unification, processus par lequel le programme logique met en correspondance un atome avec un fait ou la tête d'une règle. Ce procédé essaie de rendre deux formules identiques en donnant des valeurs aux variables qu'elles contiennent. Le résultat d'une unification est un unificateur (ou substitution), c'est-à-dire un ensemble d'affectations de variables. Si l'unification échoue, la démonstration de la formule considérée échoue. Si l'unification réussit, on substitue les variables présentes dans le corps de la clause par les valeurs correspondantes des variables de l'unificateur.

La recherche d'une preuve que la requête est une relation qui existe peut être représentée à l'aide de la procédure suivante :

prouver(requête) :

- Si un fait concorde avec la requête : on a trouvé une preuve ;
- Si une tête de règle concorde : pour chaque sous-but de cette règle, on cherche une preuve pour celui-ci à l'aide de *prouver(sous-but)*. Si tous les sous-buts sont prouvés, on a une preuve.
- Si aucun fait ou tête de règle ne concorde avec la requête, il n'existe pas de preuve.

En reprenant l'exemple donné ci-dessus, la recherche d'une preuve pour la requête `-? long_cou(X)` trouvera que le fait `long_cou(nestor)` concorde. On a donc comme solution la substitution : {`X \ nestor`}. Un programme logique peut être comparé avec une base de donnée relationnelle. L'ensemble des faits et des règles constitue les données de la base de données et les questions sont comparables à une requête sur la base de données.

Nous allons maintenant illustrer par un petit exemple la recherche d'une réponse à une requête. Pour cela, nous allons compléter les exemples donnés ci-dessus afin d'obtenir le programme suivant :

```
longues_pattes(nestor).
long_cou(nestor).
taches_sombres(nestor).
rayures_noires(nestor).

mammifere(X) :- poils(X).
mammifere(X) :- allaite(X).

carnivore(X) :- dents_pointues(X), griffes(X), yeux_en_avant(X).
carnivore(X) :- mange_viande(X).

oiseau(X) :- plumes(X).
```

```
oiseau(X) :- vole(X), oeufs(X).

espece(X,guepard) :- taches_sombres(X), couleur_fauve(X), mammifere(X),
                     carnivore(X).

espece(X,girafe) :- taches_sombres(X), longues_pattes(X), long_cou(X),
                    rayures_noires(X).

espece(X,pingouin) :- oiseau(X), sait_pas_voler(X), nage(X),
                      noir_et_blanc(X).
```

L'ensemble de toutes les déductions possibles pour une requête donnée peut se représenter sous forme d'un arbre. Diverses stratégies sont possibles pour parcourir l'arbre de recherche afin de chercher une solution. En programmation logique, les règles sont considérées dans l'ordre d'apparition dans le programme, c'est-à-dire de haut en bas et les sous-buts constituant le corps d'une règle sont considérés de gauche à droite. L'arbre est construit de manière à refléter cet ordre de considération des règles. Le mécanisme de recherche d'une preuve peut alors être exprimé en tenant compte de cette stratégie de la manière suivante. Si le sous-but le plus à gauche du but courant (à savoir g_1 dans le schéma 5.1) s'unifie avec la tête d'une clause du programme (h dans le schéma 5.1), alors il est remplacé par le corps de la clause h ($b_1, b_2, \ldots, b_n$ dans le schéma 5.1).

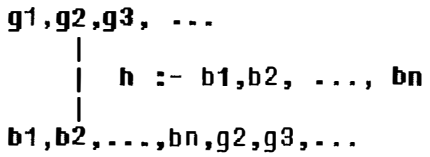


FIG. 5.1: Dérivation d'un but.

Quand le sous-but le plus à gauche a été unifié avec un fait du programme, on

peut l'effacer. Si tous les sous-buts peuvent être effacés dans un chemin de l'arbre, alors une réponse a été trouvée. Soit le programme répondra "yes", soit on a trouvé une substitution pour toutes les variables apparaissant dans la requête. La figure 5.2 montre un arbre de dérivation complet pour la requête `?-espece(nestor,X).`
Cette requête peut être traduite de la manière suivante : "de quelle espèce est Nestor ?"

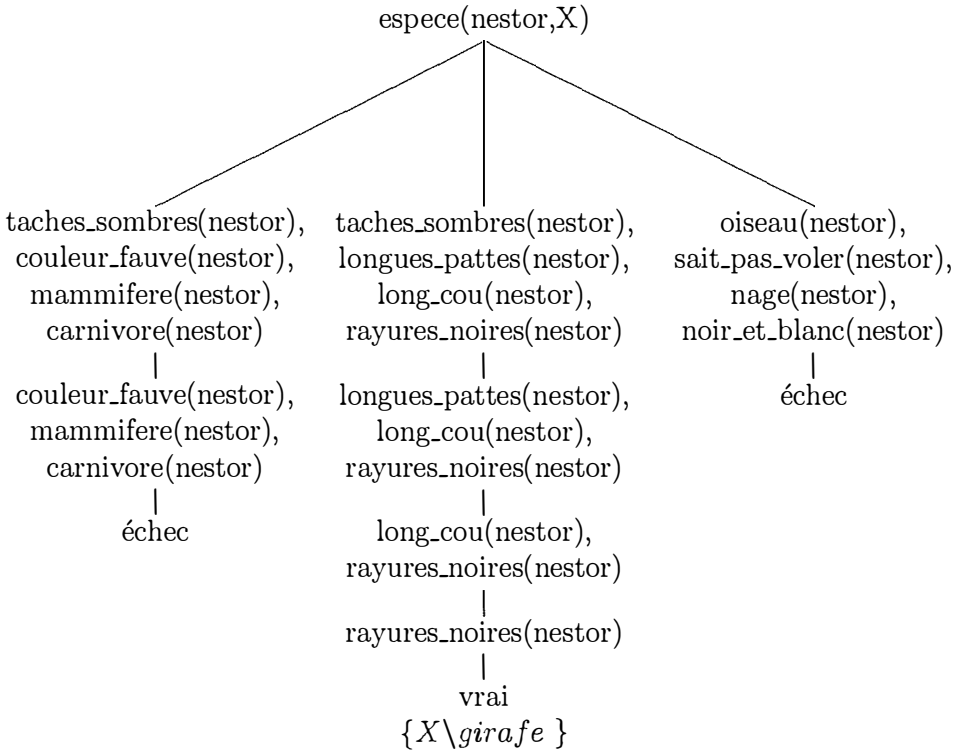


FIG. 5.2: Arbre de dérivation.

La racine de l'arbre est constituée par la requête. Chaque noeud de l'arbre correspond à un but et est composé d'une séquence de sous-buts. Les branches directement en dessous d'un noeud correspondent aux choix possibles pour remplacer le sous-but sélectionné par les règles dont la tête concordent avec ce sous-but. Ces endroits sont appelés point de choix. Par exemple, sur la figure 5.2, nous avons un point de choix à la racine puisqu'il existe trois règles dans le programme dont la tête concordent avec

la requête. Quand la branche la plus à gauche parmi celles non encore explorées aura été complètement parcourue (c'est-à-dire qu'on a soit trouvé une solution soit qu'on a échoué) le système reviendra au point de choix le plus proche afin d'explorer les autres choix encore en suspens. Ce processus est appelé "backtracking" ou retour en arrière. Par exemple, dans la figure 5.2, la branche "guepard" sera explorée puisqu'elle est la première des trois règles dans le programme. Son sous-but le plus à gauche, à savoir `taches_sombres(nestor)` concorde avec le fait `taches_sombres(nestor)` présent dans le programme. Par contre, aucun fait ne concorde avec le sous-but suivant, `couleur_fauve(nestor)`. Cet échec provoque alors un retour en arrière à l'unique point de choix et la recherche continue en explorant alors la branche "girafe" correspondant à la deuxième règle dans le programme. Le retour en arrière permet de ne pas parcourir tout l'arbre, et donc d'élaguer des branches, améliorant de cette manière les performances. Il est possible d'interdire ce retour en arrière dans certains cas comme, par exemple, lorsqu'on ne cherche qu'une seule solution et qu'on l'a trouvée. Par exemple, on pourrait interdire l'exploration de la branche "pingouin" puisqu'une solution existe dans la branche précédente.

Selon la stratégie appliquée, la manière dont les règles sont exprimées peut avoir de l'influence sur les performances. En effet, il est possible d'obtenir des boucles infinies lorsqu'on utilise des règles récursives. Ces boucles infinies conduiront le programme à ne jamais s'arrêter et donc fournir de réponse.
Mais l'expression du problème peut également amener à générer des solutions redondantes engendrant une perte de temps et une perte de capacité de mémoire dans le cas où toutes les solutions seraient recherchées. Donc, une attention particulière doit être apportée à la modélisation du problème afin de garantir au programme une efficacité optimale.

Actuellement, une des extensions les plus prometteuses de la programmation logique est la programmation logique par contraintes.

5.4 La programmation logique avec contraintes

"Constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming : the user states the problem, the computer solves it". [8]

La programmation logique avec contraintes (Constraint Logic Programming,CLP) est une généralisation de la programmation logique et consiste en l'étude de systèmes de calcul basés sur les contraintes. La programmation logique a montré ses limites en ce qui concerne le traitement d'opérations arithmétiques. En effet, en programmation logique, l'arithmétique est non déclarative, ce qui signifie que les symboles '+', '-', '=', ... ne sont pas considérés comme des symboles valides de fonction et par conséquent toute unification échoue. Par exemple, $X = Y+1$ s'exprime X `is` $Y+1$ et cela ne fonctionne que si Y est déjà fixé à une valeur numérique mais pas si X est déjà fixé à une valeur numérique et encore moins si X et Y sont libres. Donc, aucune variable non instanciée ne peut apparaître dans l'expression au moment son évaluation. C'est exactement comme l'évaluation arithmétique dans un langage procédural et cela se passe de la même manière en programmation logique avec contraintes. Afin de tenir compte de ces inconvénients, le mécanisme d'unification a été remplacé par un nouveau mécanisme, la résolution de contraintes sur un domaine particulier. Selon le domaine de satisfaction des contraintes X, on distingue plusieurs tendances notées CLP(X). Parmi ces domaines, on peut citer les entiers, les réels, les booléens, ... La résolution de contrainte arithmétique peut faire face à des égalités et des inégalités impliquant des variables pour lesquelles on essaie de trouver des valeurs permettant de satisfaire ces contraintes.

On définit une contrainte primitive comme étant une relation avec des arguments. Par exemple, $X \geq 4$ ou $X + 2Y = 9$ sont des contraintes primitives.
Une contrainte est, quant à elle, une conjonction de contraintes primitives. Par exemple, $X \geq 4 \wedge X+2Y = 9$ est une contrainte. La modélisation d'un problème consiste à formuler celui-ci en terme d'inconnues soumises à une contrainte. Cette dernière permet

de modéliser les relations entre les inconnues. De façon plus générale, une contrainte se présente sous forme d'une formule du premier ordre faisant intervenir des opérations et des relations définies dans un domaine donné. Celle-ci ne doit pas nécessairement être mathématique et résoudre la contrainte revient à trouver les valeurs qu'il faut attribuer aux variables pour la rendre vraie. Les contraintes permettent donc de décrire le comportement des objets dans le monde réel. Le problème à résoudre est aussi appelé problème de satisfaction de contraintes (Constraint Satisfaction Problem, CSP). Un CSP consiste donc en

- une contrainte C sur des variables $X_1, \ldots, X_n$;
- un domaine D qui fait correspondre à chaque variable X_i un ensemble de valeurs possibles $D(X_i)$.

Résoudre un tel problème, c'est l'objectif d'un résolveur de contraintes. Il essaie de donner des valeurs aux variables telles que la contrainte soit satisfaite. Un résolveur est une fonction qui prend comme argument une contrainte et renvoie vrai ou faux selon que la contrainte est satisfaite ou non. Une contrainte est satisfaite s'il est possible de trouver des valeurs dans le domaine des variables telles que la contrainte est vérifiée. Une assignation de valeur aux variables est appelée une valuation et une valuation qui satisfait aux contraintes est appelé une solution. Le moyen le plus simple de résoudre un CSP est d'énumérer toutes les solutions possibles en utilisant la méthode du "Generate and Test" mais cela n'est pas très efficace. Comme en programmation logique, l'utilisateur n'a pas besoin de programmer lui-même la procédure de recherche. Il peut se concentrer sur la modélisation du problème.

5.4.1 Complexité des problèmes

La complexité d'un problème et l'efficacité du programme écrit pour le résoudre dépendent très fort de la modélisation retenue. Il est souvent possible de représenter un même problème en utilisant des ensembles de variables différents et des domaines

différents. L'espace de recherche, représentant l'ensemble de toutes les valuations possibles sera différent selon le modèle retenu car il croît exponentiellement avec la taille du problème selon l'équation

$$TailleEspaceRecherche = \prod_{i=1}^{NbreDeVariables} TailleDomaine_i$$

La solution idéale serait d'avoir l'espace de recherche le plus petit possible. Par exemple, si on peut modéliser un problème à l'aide de cinq variables de domaine 1..10 et avec vingt variables de domaine 0..1, la première modélisation sera la meilleure car l'espace de recherche sera de $10^5 = 100.000$ éléments, ce qui est inférieur à $2^{20} = 1.048.576$ éléments si on opte pour le deuxième modèle. De plus, si les domaines des variables sont imposants, l'arbre de recherche construit sera gigantesque et la recherche d'une solution pourra prendre énormément de temps. Toutefois, nous avons vu dans notre cas que l'utilisateur ne cherchait à connaître qu'une seule solution à son problème, ce qui en réduit la complexité. Un tel problème est appelé problème d'existence combinatoire. Le problème d'existence combinatoire consiste donc à ne rechercher qu'une seule solution, la première trouvée étant satisfaisante, par opposition au problème d'optimisation combinatoire dont le but est de trouver parmi toutes les solutions possibles au problème la meilleure solution. Le problème d'optimisation combinatoire oblige donc à parcourir l'entièreté de l'arbre de recherche, ce qui n'est pas nécessaire pour le problème de l'existence combinatoire. Il n'est pas possible de rechercher toutes les solutions car l'arbre de recherche généré sera trop important sauf pour les petits problèmes.

La sélection des variables est importante dans la stratégie de recherche car elle peut avoir une influence sur la topologie, et donc sur la taille, de l'arbre de recherche. Pour garder une taille la plus petite possible, il est impératif de sélectionner en premier les variables dont les domaines sont les plus petits. La sélection des valeurs de variables peut aussi avoir de l'importance dans certains cas. Différents ouvrages traitent de ces stratégies, notamment [9].

Les contraintes peuvent servir à éliminer de grandes régions de l'espace qui ne contiennent pas de solutions. En général, les contraintes ne sont pas suffisamment

contraignantes pour pouvoir inférer une solution du problème. De ce fait, une phase de recherche est nécessaire afin de trouver une solution au problème. On a donc toujours deux phases dans l'exécution d'un programme :

1ère phase : Propagation de contraintes

Dans cette phase, les contraintes du problème sont utilisées afin de diminuer autant que possible les domaines des variables en éliminant les valeurs ne menant pas à une solution consistante, à l'aide de la propagation. Par exemple, supposons les variables X de domaine [1..10] et Y de domaine [4..8] et les contraintes suivantes : $X \geq 6$ et $X \leq Y$. Après la propagation de contraintes, les domaines seront réduits à [6..10] pour la variable X puisque $X \geq 6$ et à [6..8] pour la variable Y puisque $X \leq Y$ et $X \geq 6$ donc $6 \leq X \leq Y$.

2ème phase : Génération de solutions

Dans cette phase, le résolveur énumère les solutions possibles en attribuant à chaque variable une valeur appartenant à son domaine et telle que les contraintes soient respectées. Cette technique est souvent appelée "labeling". Par exemple, en reprenant l'exemple ci-dessus, X=6 et Y=6 est une solution possible.

5.4.2 Comparaison à l'aide d'un exemple : SEND+MORE=MONEY

Nous allons maintenant donner un exemple classique de la programmation logique avec contraintes, le SEND + MORE = MONEY, et le comparer avec sa solution en programmation logique. Ce problème consiste à attribuer à chaque lettre {S, E, N, D, M, O, R, Y} une valeur unique comprise entre 0 et 9 de façon à ce que l'équation SEND+MORE=MONEY soit satisfaite.

Le moyen le plus simple de modéliser ce problème est de concevoir une contrainte

correspondant à l'équation

$$\begin{array}{rl} & \mathbf{S}*1.000 + \mathbf{E}*100 + \mathbf{N}*10 + \mathbf{D} \\ & \mathbf{M}*1.000 + \mathbf{O}*100 + \mathbf{R}*10 + \mathbf{E} \\ \hline = \mathbf{M}*10.000 + & \mathbf{O}*1.000 + \mathbf{N}*100 + \mathbf{E}*10 + \mathbf{Y} \end{array}$$

Il est évident que S et M doivent être différent de 0.

En programmation logique - "Generate and Test"

La modélisation de ce problème consiste à désigner chacune des lettres par une variable, à définir le domaine de ces variables, à savoir {0, ..., 9} et à imposer les contraintes que sont l'équation et le fait que S et M doivent être différents de 0.

```
smm :- %  'déclaration' des variables
       X = [S,E,N,D,M,O,R,Y],
       %  définition du domaine
       Digits = [0,1,2,3,4,5,6,7,8,9],
       %  recherche de valeurs pour chaque variable
       assign_digits(X, Digits),
       %  vérification des contraintes
       M > 0,
       S > 0,
                 1000*S + 100*E + 10*N + D +
                 1000*M + 100*O + 10*R + E =:=
       10000*M + 1000*O + 100*N + 10*E + Y,
```

```
        % écriture de la solution
        write(X).

% les deux prédicats suivants permettent d'assigner
% une valeur à chaque variable
select(X, [X|R], R).
select(X, [Y|Xs], [Y|Ys]):- select(X, Xs,Ys).

assign_digits([], _List).
assign_digits([D|Ds], List):- select(D, List, NewList),
                              assign_digits(Ds, NewList).
```

Une solution en programmation logique de ce problème est typique du paradigme "Generate and Test". Des valeurs sont assignées aux variables et le système vérifie si les contraintes sont vérifiées. Si une des contraintes n'est pas vérifiée, le système effectue un "retour en arrière" (backtracking) pour trouver de nouvelles valeurs à assigner aux variables. Mais cela prend beaucoup de temps, d'autant plus si le problème implique beaucoup de variables et que leurs domaines sont vastes. La programmation logique n'est donc pas adéquate pour résoudre ce genre de problème notamment parce qu'elle n'est pas capable d'exploiter les contraintes définies afin de réduire l'espace de recherche.
Sans surprise, ce programme n'est pas très efficace. Le problème du SEND+MORE=MONEY exécuté à l'aide de SWI-Prolog (version 3.3.10) sur un ordinateur équipé d'un processeur PIV 1.6Ghz et doté de 512MB de mémoire nécessite une recherche de près de 15 secondes avant de fournir la réponse. Il est possible d'écrire des programmes logiques capables de réaliser de bonnes performances mais cela demandera un effort en temps et de bonnes connaissances pour les écrire.

En programmation logique avec contrainte

Avec la programmation logique avec contraintes, les contraintes seront utilisées afin de réduire l'espace de recherche. La modélisation du problème est semblable à celle réalisée en programmation logique.

```
smm(Var):- %  déclaration des variables
           Var = [S,E,N,D,M,O,R,Y],
           %  définition du domaine des variables
           Var :: [0..9],
           %  toutes les variables doivent être différentes
           alldifferent(Var),
           %  S doit être différent de 0
           S #\= 0 ,
           %  M doit être différent de 0
           M #\= 0 ,
           %  l'équation = contrainte
                       S*1000 + E*100 + N*10 + D
           +           M*1000 + O*100 + R*10 + E
           #= M*10000 + O*1000 + N*100 + E*10 + Y,
           %  recherche des valeurs
           labeling(Var).
```

Les contraintes permettront, avant de commencer la procédure de recherche, de déterminer les valeurs des variables M, S et O. En effet, la variable M devant être différente de 0 à cause de `MORE` et représentant le dernier report dans `MONEY`, elle ne peut valoir que 1. Ensuite, les valeurs de S et O peuvent être calculée grâce à S*1.000 + M*1.000 = O*1.000 + M*10.000. Comme M=1, on a que S*1.000 + 1*1.000 = O*1.000 + 1*10.000 et que S et O doivent être différent de 1. La seule valeur possible pour S permettant de résoudre cette dernière équation est 9 et O devra être égal à 0. Nous avons donc, grâce à la propagation, que S=9, M=1, O=0, ainsi que E={2..8},

N={2..8}, D={2..8}, R={2..8}, Y={2..8}.
Le prédicat `labeling(Var)`, fourni par la librairie utilisée, a pour objectif d'instancier les variables de la liste `Var` à une valeur de leurs domaines. Il joue à peu près le même rôle que les prédicats `select/3` et `assign_digits/2` dans le programme en programmation logique. La recherche d'une solution pour ce programme à l'aide du langage ECL^iPS^e prend maintenant moins d'une seconde sur le même ordinateur. Nous voyons grâce à cet exemple l'intérêt de la programmation logique avec contraintes par rapport à la programmation logique classique.

Il existe actuellement sur le marché différents systèmes de programmation logique avec contraintes, parmi lesquels GNU Prolog, Prolog III, Prolog IV, ECL^iPS^e, ILOG, CHIP, ... Leurs syntaxes sont souvent compatibles avec celle de Prolog et divergent peu entre-elles. Les différences entre ces différents outils résident principalement dans la diversité des librairies proposées. Une librairie offre différents outils afin de développer une application. En général, une librairie est spécialisée sur un domaine particulier (booléens, entiers, réels, ...)

5.5 Le langage ECL^iPS^e

5.5.1 Introduction

Le langage ECL^iPS^e est un système basé sur le paradigme de la programmation avec contraintes destiné au développement et au déploiement d'applications dans des domaines tels que la planification, l'allocation de ressources, l'établissement d'horaire, ... Il a été initialement développé à l'European Computer-Industry Research (ECRC) de Munich et est actuellement maintenu et développé par le "Centre for Planning and Resource Control" de l'Imperial College de Londres (IC-Parc).

Une interface graphique interactive, `tkeclipse`, est fournie avec le système ECL^iPS^e facilitant ainsi sa prise en main. ECL^iPS^e offre également une syntaxe commune pour

les principales contraintes fournies par différent résolveur de contraintes. Le comportement de la contrainte dépendra de la librairie utilisée. Cela permet notamment de pouvoir utiliser différents résolveurs pour résoudre une même contrainte. Mais tous les résolveurs ne supportent pas tous les types de contraintes.

5.5.2 Caractéristiques du langage

ECL^iPS^e est largement compatible avec le langage Prolog et possède des extensions significatives facilitant la modélisation des problèmes telles que le support des tableaux et des structures, des boucles, ... En effet, beaucoup d'itérations simples sont difficiles à écrire sous forme de prédicats récursifs et des constructions d'itérations logiques ressemblant fortement aux boucles que l'on rencontre dans les langages procéduraux permettent de dépasser ces difficultés.

ECL^iPS^e offre également la possibilité d'utiliser des programmes rédigés à l'origine pour un système différent sans devoir les modifier. Il est donc capable de reconnaître des dialectes du langage Prolog et d'émuler le comportement de ces autres systèmes. Les dialectes reconnus sont les suivants :

- ISO Standard Prolog ;
- C-Prolog ;
- Quintus Prolog ;
- SICStus Prolog.

Il possède également divers types de données de base tels que les réels (de simple ou de double précision), les entiers (de précision non-limitée), les nombres rationnels et les strings.

5.5.3 Les librairies disponibles

Le langage ECL^iPS^e fournit différentes librairies de résolveurs de contraintes. Ces librairies mettent à disposition des prédicats et algorithmes de recherche permettant de résoudre des problèmes variés comme la planification de tâches, les problèmes combinatoires, ...Chacune est spécialisée dans un domaine particulier. Il ne nous

est pas possible de décrire ici toutes les librairies offertes. Nous n'en décrirons que quelques-unes. Pour de plus amples informations, le lecteur peut se reporter à la documentation d'ECL^iPS^e.

Suspend

Tous les résolveurs de contraintes offert par ECL^iPS^e sont implantés en utilisant les buts suspendus. Cela consiste à suspendre le test d'une contrainte tant que toutes ses variables ne sont pas instanciées. Lorsque les variables sont instanciées, le résolveur peut alors réactiver la contrainte et vérifier si elle est satisfaite.

Domaines finis

La plupart des systèmes de programmation avec contraintes offrent ce type de résolveur, lequel applique les techniques de propagation de contraintes développées dans la communauté de l'intelligence artificielle. Cette librairie implante les domaines finis d'entiers, les fonctions et les contraintes usuelles sur les variables de ces domaines. ECL^iPS^e offre également la possibilité de travailler avec des domaines finis de valeurs non-numérique, comme par exemple "`rouge`", "`vert`", "`bleu`". De nouvelles contraintes ont aussi été insérée dans cette librairie en plus des contraintes standards supportées par les différents système CLP et elles peuvent être utilisées dans des applications spécifiques. Par exemple, ECL^iPS^e offre des librairies basées sur les domaines finis dédiées au développement d'application d'ordonnancement (scheduling).

Une contrainte est soit une contrainte arithmétique soit une combinaison de contraintes utilisant les connecteurs logiques définis sur les domaines finis ($\neg$, $\vee$, $\wedge$, $\Rightarrow$, $\Leftrightarrow$).

Intervalles

En plus des domaines finis, ECL^iPS^e offre aussi des domaines continus sous formes d'intervalles numériques.

Cette librairie associe simplement un domaine à une variable, enregistrant ses bornes inférieure et supérieure, et vérifiant le domaine quand la variable est instanciée. Elle offre également la possibilité d'utiliser des contraintes numériques utilisant le raisonnement sur intervalles. Enfin, elle implante des équations et inéquations entre des termes impliquant des variables d'intervalles continus. Cette librairie offre également des techniques de recherche pour résoudre de tels problèmes.

5.5.4 Fonctionnement

Nous allons essayer de comprendre le fonctionnement d'un résolveur en programmation logique avec contraintes. Pour cela, nous allons utiliser un petit exemple. Supposons le programme suivant :

```
:- lib(fd).

p :- q, r.
q.
r :- q.
```

Lorsqu'il compile le code source, le compilateur distingue les clauses et les directives. Les directives sont des termes utilisant le prédicat principal `:-/1` ou `?-/1`, comme par exemple `:-lib(fd)`. Lorsque le compilateur les rencontre, il exécute immédiatement leurs premiers arguments comme un but Prolog. Si ce but réussit, le compilateur continue vers le terme suivant. Ce processus est transparent pour l'utilisateur sauf si la directive échoue, auquel cas un événement est soulevé. Tous les autres termes sont interprétés comme des clauses devant être compilées. Une séquence de clauses consécutives dont les têtes sont identiques est interprétée comme une procédure. Si les clauses dont les têtes sont identiques ne sont pas consécutives, l'utilisateur est averti que la dernière procédure définie remplace la précédente.

L'ensemble de tous les buts qui doivent être satisfais sont représentés par ce que l'on nomme, en programmation logique, le "resolvent". L'exécution d'un programme est déclenchée par une requête et la recherche d'une solution est semblable à celle utilisée en programmation logique et explicitée à la section 5.3 de ce chapitre. Par exemple, la requête `-?p.` déclenchera l'exécution et le "resolvent" passera par les états suivants avant que la requête ne soit prouvée : p $\dashrightarrow$ q,r $\dashrightarrow$ r $\rightarrow$ q $\rightarrow$ {}

ECL^iPS^e possède un résolveur nommé "suspend" fournissant une règle de calcul plus puissante que le calcul standard gauche-droite, utilisé notamment par Prolog. Au lieu de traiter le "resolvent" de gauche à droite comme en Prolog, ECL^iPS^e peut le manipuler de manière plus flexible car le "resolvent" est plus structuré. Cette gestion du "resolvent" sera effectuée par un "waking scheduler" et elle est rendue possible grâce à deux mécanismes de base : la suspension et les priorités.

La priorité ECL^iPS^e supporte, dans sa version 5.2, douze priorités différentes attribuées aux buts, la priorité 1 étant la plus urgente et la priorité 12 étant la moins urgente. La figure 5.3 montre la structure du "resolvent". Le mécanisme de déduction s'enclenche lorsqu'une requête est introduite. Celle-ci sera introduite dans le "resolvent", dont elle sera le seul membre, avec une priorité 12. Au fur et à mesure que l'exécution se poursuit, les buts peuvent être suspendus lorsqu'on constate qu'aucune information intéressante ne pourra être inférée de ceux-ci et les buts suspendus peuvent être réactivés et se voir affecter une priorité particulière. Les buts restants sont ordonnés selon leurs priorités. Le système tente d'abord de résoudre le sous-but le plus urgent. La suspension est un type de données opaque utilisé pour représenter les buts suspendus. Un but suspendu attend généralement que certaines conditions soient réunies afin d'être réactivé. Quand les conditions sont réunies, la suspension est passée à un "waking scheduler" qui la placera dans la file d'attente des buts réveillés et dès qu'elle devient la première de la file, la suspension est exécutée. Cette file est ordonnée selon les priorités accordées. L'événement qui fait qu'une suspension sera réveillée est généralement apparenté à une ou plusieurs variables, par

exemple à une instanciation de variable.

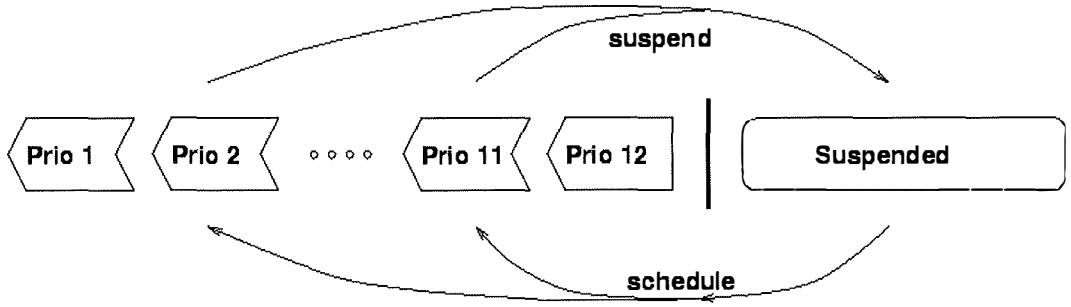


FIG. 5.3: Structure du "resolvent".

La suspension Une suspension contient des informations nécessaires pour contrôler l'exécution du but suspendu qu'elle représente, notamment la structure du but, le module à partir duquel le but a été appelé, la priorité qui sera attribuée au but lorsqu'il sera réactivé ou l'état du but. Il est soit "suspendu", "planifié" ou "exécuté". Une suspension qui n'a pas encore été exécutée ou planifiée pour l'exécution est dite "endormie", une suspension qui a déjà été exécutée est dite "exécutée" et disparaît du "resolvent". Une suspension exécutée peut redevenir endormie à cause du backtracking.

5.5.5 La communication entre ECL^iPS^e et d'autres langages

Les langages de programmation courants comme Java, C ou C++ sont souvent préférés dans les entreprises aux langages spécialisés, notamment parce qu'il est plus facile de trouver un informaticien connaissant ces langages. C'est pourquoi il est possible de faire appel à partir du code ECL^iPS^e à du code C/C++ et inversément. Les données peuvent aussi être converties entre la représentation en C et la représentation en ECL^iPS^e. Il est également possible de faire appel à des fonctions codées en ECL^iPS^e à partir du langage Java, Tcl/Tk ou Visual Basic (VB).

5.6 Méthodes de résolution

La programmation logique semble une bonne base pour résoudre le problème posé et défini au chapitre 4. En effet, comme nous le verrons dans le chapitre suivant, les règles de regroupement se prêtent particulièrement bien à la modélisation logique tout comme les besoins définis.

Mais comme nous utilisons des expressions arithmétiques du style $X > 2$, la programmation logique n'est pas adaptée à notre problème comme l'exemple du SEND+MORE=MONEY nous l'a montré. En effet, la présence de ces expressions, mais aussi le grand nombre de variables et leurs domaines assez conséquents nous oblige à chercher d'autres méthodes que celle du "Generate and Test" telle qu'on peut l'appliquer en programmation logique. La programmation logique avec contraintes offre des solutions pour surmonter ces difficultés.

Le langage ECL^iPS^e offrant un grand nombre de librairies, nous avons préféré celui-ci afin de développer notre prototype. Toutefois, d'autres langages auraient pu se prêter à cet exercice. En effet, le domaine des variables utilisées pour la modélisation des différents besoins identifiés au chapitre 4 peut être considéré comme finis. Or la librairie offrant la possibilité de travailler avec des variables de domaines finis est fournie par la majorité des langages de programmation avec contraintes. Seules certaines variables possèdent des domaines continus. Mais les domaines continus ne s'appliquent que pour des variables numériques, ce qui n'est pas le cas de toutes les variables du problème et la librairie permettant de travailler sur de telles variables n'offrent pas la possibilité d'utiliser les connecteurs logique $\wedge$ et $\vee$. C'est pourquoi nous avons adapté ces domaines de façon à pouvoir les exprimer à l'aide de domaines finis.

Chapitre 6

Application de la programmation avec contraintes à notre problème

6.1 Introduction

Dans ce chapitre, nous verrons tout d'abord l'architecture du prototype développé afin de résoudre le problème spécifié au chapitre 4. Ensuite nous verrons la modélisation effectuée afin de pouvoir réaliser les validations de règles et, enfin, nous aborderons les résultats obtenus à l'aide de ces modèles.

6.2 Architecture du prototype

Nous allons ici expliquer le prototype développé afin de comprendre son fonctionnement. La figure 6.1 nous servira de référence.

L'éditeur de règles ayant pour objectif la gestion des règles de regroupement définies par les comptables, nous avons besoin de stocker ces règles, ce à quoi servira le fichier `Regles.xml`. Toutefois, les règles y seront stockées exactement comme l'utilisateur les aura définies. Cette remarque vaut notamment pour la définition de règles à l'aide de la fonctionnalité de complémentaire. En effet, les complémentaires

exprimés par l'utilisateur ne seront pas traduit directement. Par exemple, si l'utilisateur demande le calcul du complémentaire d'une expression booléenne $X < 2$, nous ne sauverons pas dans le fichier le complémentaire, à savoir $X \geq 2$, mais plutôt l'expression `complémentaire(`$X < 2$`)`. Le fichier `ReglesComplet.xml` contiendra ,lui, les règles avec les complémentaires calculés. La décision de garder l'expression des complémentaires dans le fichier `Regles.xml` est motivée par le fait qu'on doit être capable d'indiquer à l'utilisateur, lorsqu'il réalise une importation des règles dans l'éditeur, les expressions booléennes, sous-règles ou règles qui sont exprimées comme le complémentaire d'autres expressions booléennes, sous-règles ou règles.

Nous avons vu que l'éditeur de règles devait être convivial. Pour cela, nous avons besoin d'accéder à la base de données de la banque afin de récupérer certaines informations. Cet accès permettra également de calculer les domaines des variables, nécessaires pour réaliser la validation intelligente des règles.

Mais nous savons que les performances en programmation logique avec contraintes dépendent du nombre de variables intervenant dans le modèle du problème ainsi que de la taille de leurs domaines. En l'occurrence, dans notre problème, près de trente variables ont été recensées. De plus, les contraintes spécifiées peuvent parfois se révéler insuffisamment contraignantes pour réduire l'espace de recherche de manière significative. Mais la totalité des domaines n'étant pas toujours nécessaire à la résolution d'un problème, il a été décidé, et ce pour améliorer les performances, de laisser la possibilité de calculer le domaine de chaque variable de deux manières. Soit on calcule complètement le domaine à l'aide d'une requête SQL sur la base de données soit on définit "manuellement" un sous-ensemble du domaine réel en énumérant les valeurs de la variable concernée. Ces informations, à savoir la requête ou l'énumération des valeurs du domaine, sont stockées dans le fichier `Requetes.xml`. Les domaines calculés, eux, seront stockés dans un fichier spécifique, `Domaines.xml`. L'accès à la base de données sera gérée par une couche spécifique, nommée `COUCHE DOMAINE`.

Un désavantage de la programmation logique avec contraintes est que lorsqu'aucune solution ne peut être trouvée à un problème, le système ne donne aucune explications à l'utilisateur sur les raisons de cet échec. La partie `EXPERT` de l'éditeur de règles permet de chercher ces raisons. Toutefois, il faut remarquer que des chercheurs se sont orientés dans cette voie afin de fournir une mécanisme d'explications des échecs de recherche de solutions. C'est ce que l'on nomme la programmation par contraintes avec explications. Ces recherches n'en est encore qu'à leurs débuts et donc très peu de systèmes offrent ce mécanisme.

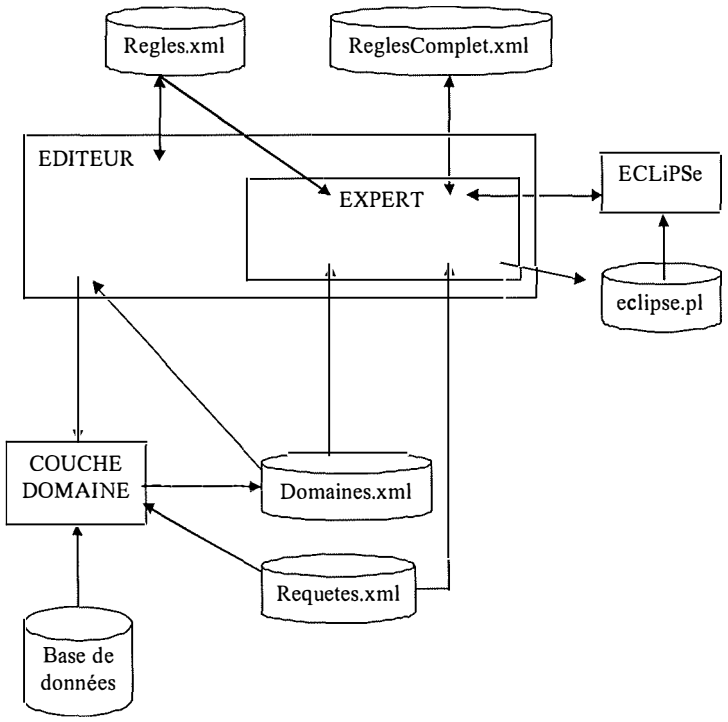


FIG. 6.1: Architecture du prototype.

La demande de validation de règles provoquera la création d'un fichier contenant le code nécessaire à ECL^iPS^e pour effectuer les vérifications. Les domaines des variables utilisées seront récupérées dans le fichier `Domaines.xml` tandis que les éléments nécessaires (expressions booléennes, sous-règles, règles) seront puisées dans le fichier

`ReglesComplet.xml`. La partie `Expert` de l'outil de gestion de règles pourra alors demander l'exécution du programme contenu dans le fichier `eclipse.pl` par ECL^iPS^e et récupérera la solution fournie. La partie Expert n'aura plus qu'à traiter cette solution et soit avertir l'utilisateur en cas de problème soit continuer ses recherches.

Le lecteur est invité à se reporter à l'annexe C pour des explications concernant la structure des fichiers XML utilisés et pour découvrir des extraits de ceux-ci.

6.3 Modélisation des règles de regroupement

La définition des règles de regroupement donnée au chapitre 4 facilite la modélisation en programmation logique avec contraintes. En effet, une expression booléenne peut être assimilée à une contrainte primitive et une sous-règle à une contrainte. Une règle étant constituée de la disjonction de sous-règles, on la représente donc comme une disjonction de contraintes. Nous représentons une classification de la même manière, à savoir comme une disjonction de règles. En supposant les indices
i $\in \{1..n\}$ représentant le numéro de la classification CL ;
j $\in \{1..m\}$ représentant le numéro de la règle R dans la classification CL_i ;
k $\in \{1..p\}$ représentant le numéro de la sous-règle SR dans la règle R_{ij} ;
l $\in \{1..q\}$ représentant le numéro de l'expression booléenne EB dans la sous-règle SR_{ijk}, nous pouvons formaliser cette modélisation comme suit :

$$EB_{ijkl} \equiv < variable >< operateur >< argument >$$
$$SR_{ijk} \equiv EB_{ijk1}\ ET \ldots ET\ EB_{ijkl}$$
$$R_{ij} \equiv SR_{ij1}\ OU \ldots OU\ SR_{ijk}$$
$$CL_i \equiv R_{i1}\ OU \ldots OU\ R_{ij}$$

et l'exprimer en logique de la manière suivante :

$$EB_{ijkl} \equiv < variable >< operateur >< argument >$$

$SR_{ijk} \equiv EB_{ijk1} \wedge \ldots \wedge EB_{ijkl}$
$R_{ij} \equiv SR_{ij1} \vee \ldots \vee SR_{ijk}$
$CL_i \equiv R_{i1} \vee \ldots \vee R_{ij}$

Maintenant que nous savons comment représenter les règles de regroupement, nous allons pouvoir aborder la modélisation des différents besoins identifiés.

6.4 Modélisation des besoins identifiés

Pour chacun des besoins identifiés et définis dans le chapitre 4, nous donnerons une modélisation facilitant la traduction en programmation logique par contrainte.

6.4.1 La compatibilité

Nous savons par définition qu'une sous-règle est compatible si et seulement si
$\forall EB_{ijka} \in SR_{ijk} \ : S(EB_{ijka}) \neq \emptyset$
et
$\forall EB_{ijka}, EB_{ijkb} \in SR_{ijk}$ *tellesque* $a \neq b, vars(EB_{ijka}) = vars(EB_{ijkb}) \ :$
$S(EB_{ijka}) \cap S(EB_{ijkb}) \neq \emptyset$

Une sous-règle

Pour vérifier la compatibilité d'une sous-règle, nous devons tester chaque expression booléenne afin de détecter une éventuelle incompatibilité par rapport au domaine mais aussi tester deux à deux toutes les expressions booléennes partageant la même variable.

Une règle

Une règle consiste en un ensemble de sous-règles. Donc, pour déterminer si la règle est compatible ou non, il suffit de tester chacune de ses sous-règles.

Une classification

Une classification, par définition, consiste en un ensemble de règles. Pour tester sa compatibilité, il faut tester la compatibilité de chaque règle la constituant.

Les classifications

L'utilisateur peut demander de tester la compatibilité de toutes les classifications qui ont été définies. Cela consiste en fait à tester chacune de ces classifications une à une.

Principe

Lorsque nous disposons d'une sous-règle, nous vérifions sa compatibilité. S'il existe une solution vérifiant la sous-règle, nous pouvons tester la sous-règle suivante. Par contre, si aucune solution n'existe, il faut en déterminer la cause. Soit une de ses expressions booléennes est incompatible par rapport à son domaine soit deux de ses expressions booléennes sont incompatibles entre-elles.
En principe, si deux expressions booléennes partageant la même variable sont compatibles entre-elles, cela signifie en même temps qu'elles sont toutes les deux compatibles par rapport à leurs domaines respectifs. C'est pourquoi, nous testerons d'abord la compatibilité entre les expressions booléennes deux à deux. Cela devrait permettre dans certains cas d'améliorer le temps de réponse. Si la cause de l'incompatibilité n'est pas découverte de cette manière, cela signifie qu'une expression booléenne, ne partageant pas sa variable avec d'autres expressions booléennes est incompatible par rapport à son domaine.

Étant donné que l'utilisateur n'a pas exprimé le besoin de connaître toutes les causes d'incompatibilité en une fois, et aussi pour des raisons de performance, nous ne chercherons qu'une seule solution à un test de compatibilité entre deux sous-règles. Cela évite de parcourir tout l'arbre de recherche qui sera généré. En effet, une sous-règle est compatible par rapport à son domaine si son évaluation renvoie au moins une

valeur. Deux sous-règles sont compatibles entre-elles si l'évaluation de leur conjonction renvoie au moins une valeur. Il n'est donc pas nécessaire de connaître toutes les valeurs rendant les sous-règles compatibles. Une contrainte du langage ECL^iPS^e fournie par la librairie de domaines finis permet de formuler cette contrainte. Il s'agit du prédicat *once/1*.

Code ECL^iPS^e

Compatibilité d'une sous-règle contenant `k` variables et `s` expressions booléennes ($s \geq k$) :

```
:-lib(fd).

compatibilite(V):- V=[Variable_1, ..., Variable_k],
                   Variable_1::[domaine], ..., Variable_k::[domaine],
                   Expression booleenne_1
                #/\ ...
                #/\ Expression booleenne_s,
                   once labeling(V).
```

6.4.2 Le recouvrement

D'après les définitions données dans le chapitre 4, nous savons qu'il existe un recouvrement entre deux règles si elles possèdent au moins une solution commune, c'est-à-dire si $S(R_{ia}) \cap S(R_{ib}) \neq \varnothing$.

Deux règles

L'utilisateur peut demander de tester deux règles bien particulières lorsqu'il a un doute concernant la validité de celles-ci.

Une classification

Une classification constitue un ensemble de règles or le recouvrement doit être testé sur des règles, deux à deux. Tester le recouvrement sur une classification consiste donc à tester toutes ses règles deux à deux.

Les classifications

L'utilisateur peut demander la recherche de l'existence d'un recouvrement dans une classification parmi toutes celles qui ont été définies. Pour cela, il faut tester chaque classification une à une.

Principe

Nous pouvons réexprimer la définition du recouvrement que nous avons rappelée ci-dessus. Le recouvrement impose de chercher une valeur $(x_1, \ldots, x_n)$ telle que
$x_1 \in dom(X_1), \ldots, x_n \in dom(X_n)$
et $\{X_1, \ldots, X_n\} = vars(R_{ia}) \cup vars(R_{ib})$
et $R_{ia} \backslash (x_1, \ldots, x_n) = \text{true}$
et $R_{ib} \backslash (x_1, \ldots, x_n) = \text{true}\}$.
Cela signifie que l'on cherche des valeurs pour les variables apparaissant dans les règles qui permettent de vérifier en même temps les deux règles. Nous cherchons donc un n-uplet de valeurs tel que la conjonction des deux règles $R_{ia} \wedge R_{ib}$ est vérifiée.
S'il n'existe pas de solution à ce problème, cela signifie donc qu'aucun cas de recouvrement n'a été décelé entre les deux règles. Dans le cas contraire, nous avons besoin d'en connaître les causes en déterminant les deux sous-règles à l'origine de ce recouvrement. Nous savons qu'il existe un recouvrement entre deux sous-règles si elles possèdent au moins une solution commune, c'est-à-dire si
$\forall n, m \ (n \neq m) \ S(SR_{ian}) \cap S(SR_{ibm}) \neq \varnothing$. Nous testerons donc, deux à deux, les sous-règles appartenant à la ou les règles mise(s) en cause.

La recherche d'une solution commune peut demander du temps lorsque les deux règles possèdent un grand nombre de variables, pas nécessairement toutes communes, et que leurs domaines sont de taille importante. Afin de réduire ce temps de recherche, on peut limiter la recherche en ne choisissant que les variables opportunes à celle-ci, à savoir les seules variables communes aux deux règles. En effet, une variable n'appartenant qu'à une seule des deux règles aura obligatoirement une valeur commune dans les deux règles si on effectue une recherche de solution commune incluant cette variable. Nous allons illustrer ces propos par un exemple. Supposons les deux règles suivantes.

```
code_pays::[500,501],
duree::[1,2,3],
code_activite::[100,101,102],

SI code_pays=500#501 ∧  duree > 1 ∧  code_activite = 101#102
ALORS valeur = 1013000

SI code_pays=500 ∧  duree < 3
ALORS valeur = 1012000
```

Nous devons donc chercher les valeurs communes aux deux règles. Si nous cherchons pour toutes les variables, nous aurons comme solutions pour la première règle :

```
[code_pays,duree,code_activite]={(500;2;101),(500;2;102),(500;3;101),
                                 (500;3;102),(501;2;101),(501;2;102),
                                 (501;3;101),(501;3;102)}
```

et pour la seconde règle :

```
[code_pays, duree,code_activite]={(500;1;100),(500;1;101),(500;1;102),
                                  (500;2;100),(500;2;101),(500;2;102)}
```

Comme la variable `code_activite` n'apparaît pas dans l'énoncé de la seconde règle, elle prendra toutes les valeurs possibles de son domaine. Or, elle est contrainte dans

la première règle par l'expression booléenne `code_activite = 101#102`. Sa valeur est donc déterminée. Cette variable prendra donc nécessairement les mêmes valeurs que dans la première règle. Il n'est donc pas nécessaire de rechercher les valeurs des variables n'apparaissant pas simultanément dans chacune des deux règles à comparer.

Nous voyons ainsi que les solutions (500 ;2 ;101) et (500 ;2 ;102) sont communes aux deux règles. Si on ne recherche des valeurs que pour les variables communes aux deux règles, nous aurons comme ensemble de solutions pour la première règle [code_pays, duree]={(500 ;2),(500 ;3) ;(501 ;2),(501 ;3)} et pour la seconde règle [code_pays, duree]={(500 ;1),(500 ;2)}, ce qui donne comme solution commune (500 ;2). On génère donc moins de solutions en ne calculant que les solutions des variables communes et on améliore ainsi le temps de réponse.

Cette optimisation nous amène à modifier légèrement la modélisation proposée ci-dessus. Nous cherchons maintenant une valeur $(x_1, \ldots, x_n)$ telle que
$x_1 \in dom(X_1), \ldots, x_n \in dom(X_n)$
et $\{X_1, \ldots, X_n\} = vars(R_{ia}) \cap vars(R_{ib})$
et $R_{ia} =$ true
et $R_{ib} =$ true.

Nous pouvons exprimer cela en logique de la manière suivante : nous cherchons une valeur $(x_1, \ldots, x_n)$ telle que $x_1 \in dom(X_1), \ldots, x_n \in dom(X_n)$
et $\{X_1, \ldots, X_n\} = vars(R_{ia}) \cap vars(R_{ib})$
et $R_{ia} \wedge R_{ib}$.

Code ECL^iPS^e

La traduction en programmation logique avec contraintes ne présente pas de difficultés.

```
:-lib(fd).

recouvrement(V):-    V=[Variable_1, ..., Variable_k],
```

$Variable_1$::[domaine], ..., $Variable_k$::[domaine],
$Regle_1$ #/\ $Regle_2$,
once labeling(V).

6.4.3 La définition complète

Pour rappel, une classification est dite complètement définie si l'ensemble des solutions de cette classification est égal au produit cartésien des domaines de ses variables, c'est-à-dire si $S(CL_i)$ = produit cartésien des domaines.

La classification ne sera donc pas complètement définie s'il existe un n-uplet de valeurs appartenant au produit cartésien des domaines mais pas à l'ensemble des solutions de la classification, c'est-à-dire si le complémentaire de l'ensemble des solutions de la classification est non vide.

Une classification

Une valeur qui ne vérifie aucune des règles de la classification, c'est-à-dire qui n'appartient pas à l'ensemble des solutions de celle-ci est une valeur appartenant au complémentaire de cet ensemble.

Les classifications

L'utilisateur peut demander l'application de ce test sur toutes les classifications définies. Cela signifie que chaque classification devra être testée une à une.

Code ECL^iPS^e

```
:-lib(fd).
definition(V):- V=[Variable_1, ..., Variable_k],
                Variable_1::[domaine], ..., Variable_k::[domaine],
                #\+( (Regle_1) #\/ (Regle_2) #\/ ... #\/ (Regle_k) ),
                once labeling(V).
```

6.5 Résultats obtenus

Cette section nous montre l'importance de développer un module de validation des règles introduites par l'utilisateur. La rapidité avec laquelle l'application fournira un résultat dépendra aussi, en plus des aspects propres à la programmation par contraintes comme la stratégie appliquée lors de la recherche, de la localisation d'une erreur. En effet, comme nous ne cherchons qu'une erreur, la première rencontrée sera mentionnée et la recherche s'arrête. Un prototype a été développé à l'aide de la plate-forme ECL^iPS^e et du langage Java.

Pour réaliser les tests, nous avons utilisé les règles de regroupement en cours de définition que nous avons traduite en format XML. Une centaine de règles de regroupement étaient ainsi à notre disposition, faisant intervenir 8 variables différentes. Nous allons citer chaque variable et indiquer la taille de leurs domaines respectifs. Nous indiquerons aussi entre parenthèses la méthode utilisée pour calculer le domaine à savoir à l'aide d'une requête SQL ou bien à l'aide d'une énumération des valeurs :

CODE_ACTIVITE : 50 éléments (REQUÊTE)
PAYS_RISK : 261 éléments (REQUÊTE)
DUREE_R : 3501 éléments (ÉNUMÉRÉ)
W1i : 31 éléments (ÉNUMÉRÉ)
ESPECE_PORTF_CTV : 3 éléments (ÉNUMÉRÉ)
TYP_REC_POS : 7 éléments (ÉNUMÉRÉ)
TYPE_RAP : 42 éléments (REQUÊTE)
CATEGORIE_CPT : 17 éléments (ÉNUMÉRÉ)

Le produit cartésien des domaines est alors composé de pas moins de
21.236.445.272.700 éléments.

Une première ébauche du prototype permettait de récupérer toutes les solutions d'un test. Seulement cela posait des problèmes évidents lorsque l'espace de recherche était volumineux et qu'il existait beaucoup de solutions. En effet, nous étions obligé de stocker toutes les solutions en mémoire jusqu'à ce que la dernière soit trouvée

provoquant parfois un débordement de la mémoire ("stack overflow"). De plus cette recherche de toutes les solutions allonge le temps nécessaire de recherche avant d'obtenir une solution.

L'utilisateur n'ayant pas nécessairement besoin de toutes ces solutions en une fois, il a été décidé de n'en rechercher qu'une seule. Comme nous sommes face à un problème d'existence combinatoire, la taille des domaines a un peu moins d'importance. De plus, il n'y a pas de risque d'avoir une boucle infinie puisque nous ne définissons qu'un seul prédicat dans chaque fichier utilisé par ECL^iPS^e, celui-ci ne s'appelant pas récursivement. Par contre, ce qui prendra la plus grande partie du temps de la validation est la manipulation des règles de regroupement par Java afin qu'ECL^iPS^e puisse effectuer un test. En effet, pour certaines validations comme le recouvrement, nous devons tester toutes les règles deux à deux. Cela signifie donc que plus il y a de règles, plus il existera de combinaisons et donc plus le temps nécessaire pour les générer sera important.

Nous avons effectué les tests sur toutes les règles définies dans le fichier `ReglesComplet.xml`, à savoir une centaine, et contenant près de 130 sous-règles. Ces tests ont été effectués sur un ordinateur équipé d'un processeur PIV 1.6GHz et de 512 MB de mémoire. Le lecteur trouvera dans l'annexe C des tests effectués avec leurs code ECL^iPS^e ainsi que les réponses fournies à l'utilisateur par le prototype.

6.5.1 La compatibilité

Dans le meilleur des cas, à savoir si l'incompatibilité est située dans la première sous-règle testée, une réponse sera fournie après environ 3 secondes. Dans le pire des cas, c'est-à-dire si l'incompatibilité est générée par la dernière sous-règle testée, près de 7 secondes seront nécessaire avant d'obtenir une réponse. Si le nombre de règles est multiplié par 5, le temps nécessaire dans le pire des cas avoisinera les 25 secondes. Cette progression linéaire provient du fait que les règles sont testées les unes après les autres.

Dans le cadre du stage, la recherche d'incompatibilités sur les règles de regroupement déjà définies a révélé que toutes les classifications étaient compatibles.

6.5.2 Le recouvrement

Dans le meilleur des cas, c'est-à-dire que le recouvrement est provoqué par les deux premières règles testées, une réponse sera fournie en moins de 5 secondes à l'utilisateur. Dans le pire des cas, c'est-à-dire que le recouvrement est causé par les deux dernières règles testées, il faudra près de 8 minutes au système afin de déterminer la cause du recouvrement.

Le même test effectué sur un ensemble réduit de règles, 30, prendra, dans le pire des cas, environ 25 secondes au système. On voit donc ici que la progression du temps n'est plus linéaire comme c'est le cas dans le test de compatibilité. Cela provient du fait que les règles doivent être testées deux à deux. Plus le nombre de règles augmente dans une classification, plus le nombre de combinaison possible entre ces règles augmente. D'où l'évolution non linéaire du temps nécessaire pour trouver une solution.

Quelques cas de recouvrement ont été découverts dans les règles de regroupement déjà définies dans le cadre du projet du "nouveau reporting".

6.5.3 La définition complète

Le temps nécessaire pour trouver une réponse à ce problème sera en grande partie consacrée à la création du fichier `eclipse.pl`. En effet, plus il y aura de règles dans une classification, plus le traitement des règles afin de créer la contrainte prendra du temps.

Dans le pire des cas, le temps nécessaire avant d'obtenir une réponse avoisinera les 10 secondes. Il est à remarquer qu'ECL^iPS^e ne prend même pas une seconde pour répondre. Lorsque le nombre de règles augmente, par exemple à 200, le temps nécessaire sera de 30 secondes et de 1 minutes pour 300 règles.

Des classifications non complètement définies ont aussi été détectées dans les règles de regroupement déjà définies dans le cadre du projet du "nouveau reporting".

Conclusion

Dans ce travail, nous avons étudié le problème de la validation intelligente de données. Dans notre cas, la validation concernait des règles exprimées par des comptables dans le cadre du processus de création de documents financiers selon les exigences légales. Elle se doit de détecter des erreurs d'expression de ces règles avant que celles-ci ne se propagent aux documents produits et ne soient alors plus difficile à localiser. L'objectif de ce travail était également de montrer l'intérêt de travailler avec la programmation logique avec contraintes, d'autant plus que le problème posé s'y prêtait particulièrement bien.

La programmation logique avec contraintes nous permet de modéliser des problèmes complexes et de les résoudre dans un intervalle de temps raisonnable notamment grâce à ses algorithmes de recherche intégrés ce dont ne disposent pas les langages de programmation logique comme Prolog. Son efficacité tient au fait qu'elle permet de dissocier la représentation du problème (réalisée par l'utilisateur) de sa résolution (réalisée par le système). Cette facilité de modélisation des problèmes permet une évolution constante du système afin de répondre aux nouveaux besoins de utilisateurs et également un développement plus rapide. Toutefois, les performances de ce système dépendront toujours du nombre de variables intervenant dans la modélisation du problème ainsi que de leurs domaines. Vu l'impossibilité d'appliquer, pour la résolution de notre problème, une méthode de recherche du type "Generate and Test", nous avons été obligés de chercher une méthode permettant de réduire l'espace de recherche du problème. La programmation logique avec contraintes nous offre cette possibilité ainsi qu'un gain de temps non négligeable et une intégration facile avec des

langages de programmation classique répandus dans les entreprises à l'heure actuelle comme Java, C ou C++.

Mais on a vu aussi que son principal désavantage provient de son manque d'expression des raisons d'un échec de résolution d'un problème, ce que la programmation par contraintes avec explications tente de résoudre. Un tel mécanisme permettrait probablement, dans notre cas, de gagner encore du temps en évitant des manipulations de règles à l'aide de Java. Un tel mécanisme augmenterait encore l'attractivité de la programmation logique avec contraintes. Ces recherches doivent donc être suivies avec beaucoup d'attentions.

Bibliographie

[1] Banque Nationale de Belgique, La centrale des bilans. *http ://www.bnb.be.*

[2] Commission de Surveillance du Secteur Financier, Missions et compétences. *http ://www.cssf.lu.*

[3] T. Lemlouma et A. Boudina. Programmation logique avec contraintes (clp), Étude et application au puzzle : 'send plus more equal to money'.

[4] A. Thayse et co auteurs. *Approche logique de l'intelligence artificielle, De la logique classique à la programmation logique*, volume 1. Dunod informatique, 1990.

[5] A. Aho et J. Ullman. *Concepts fondamentaux de l'informatique.* Dunod, 1992.

[6] K. Marriott et P. J. Stuckey. *Programming with Constraints : An Introduction.* MIT Press, 1998.

[7] F. Fages. *Programmation Logique par contraintes.* ellipses, 1996.

[8] Eugene C. Freuder. in CONSTRAINTS, Avr 1997.

[9] P. Van Hentenryck. *Constraint Satisfaction in Logic Programming.* MIT Press, 1989.

[10] J-M. Jacquet. *Cours de technique d'intelligence artificelle.* Facultés Universitaires Notre-Dame de la Paix, Namur.

[11] L. Sterling and E. Shapiro. *The Art of Prolog.* 1986.

[12] C. Van Wymeersch. *Comptabilité financière, Introduction au système d'information financier de l'entreprise.* Facultés Universitaires Notre-Dame de la Paix, Namur.

Annexes

A. Documents comptables : Les différents types de ventilation

B. Documents comptables : Situation active et passive

C. Les fichiers en format XML

D. Résultats obtenus

Annexe A

Les différents types de ventilation

TABLEAU S 1.1
BILAN STATISTIQUE MENSUEL

INTRODUCTION

L'objet du tableau S 1.1 est de permettre à la Banque centrale du Luxembourg (BCL) de collecter les données nécessaires à l'élaboration de statistiques monétaires et bancaires. Les données ainsi produites seront publiques et serviront notamment à la Banque centrale européenne (BCE) dans la définition de la politique monétaire commune.

Le tableau S 1.1 comprend uniquement un bilan. Les rubriques, qui sont les mêmes que celles utilisées au niveau du tableau S 2.5 "Bilan statistique trimestriel", sont obtenues en regroupant les opérations ventilées selon le pays, la devise et l'échéance dans le tableau trimestriel précité. Ces rubriques sont également des regroupements de rubriques du tableau B 1.1 «Situation active et passive», tel qu'il est décrit dans la partie III du présent recueil des instructions aux banques.

Le tableau S 1.1 devra être renseigné en version L. Il sera en outre à renseigner dans la monnaie du capital, les conversions devront se faire au cours du jour de l'établissement du tableau, conformément aux règles détaillées dans les Définitions et Commentaires Préliminaires (XVI.12.c.).

Le tableau S 1.1 est à transmettre à la BCL sous forme de fichier informatique respectant les normes EDI telles qu'elles ont été définies dans le document «Schedule of Conditions for the Technical Implementation of the CSSF / BCL Reporting Requirements».

Les différents types de ventilation

Les différents postes de l'actif et du passif devront être ventilés selon l'échéance initiale, la devise, le pays et le secteur économique de la contrepartie. Les nomenclatures et les codes sont détaillés par la suite.

1. L'échéance initiale

Ne sont à ventiler selon l'échéance initiale que les rubriques relatives aux titres de créance détenus, aux titres de créance émis, aux dettes à terme et aux dettes à préavis.

Les titres de créances détenus, les titres de créance émis ainsi que les dettes à terme sont à ventiler dans les trois classes suivantes:

- ≤ 1 an
- > 1 an et ≤ 2 ans
- > 2 ans

Les dettes à préavis sont à ventiler dans les trois classes suivantes:

- ≤ 3 mois
- > 3 mois et ≤ 2 ans
- > 2 ans

2. La devise

Les montants sont à ventiler selon la devise utilisée dans une des deux classes suivantes:

- Euro: cette classe regroupe l'ensemble des opérations libellées en euros ou dans des dénominations nationales non décimales de l'euro[1];
- Autres devises: cette classe regroupe l'ensemble des opérations libellées dans d'autres devises que l'euro ou dans des dénominations nationales non décimales de l'euro

3. Le pays

Les montants seront également à ventiler selon le pays de résidence ou du siège social de la contrepartie c'est-à-dire le pays dans lequel se situe le centre d'intérêts économiques de la

[1] Il s'agit des devises nationales actuelles des pays ayant rejoint l'Union monétaire, à savoir BEF, DEM, FRF, ESP, IEP, ITL, LUF, NLG, ATS, FIM, PTE.

contrepartie en question. Une contrepartie est à considérer comme étant résident dans un pays lorsqu'elle y a poursuivi des activités économiques pendant au moins un an.

Ce principe de territorialité, le seul pertinent pour l'analyse économique des statistiques bancaires internationales, vaut pour toutes les contreparties de l'établissement rapportant, y compris pour les établissements bancaires succursales de banques étrangères. La perspective du tableau B 1.1. est différente sur ce dernier point: le contrôle prudentiel mettant l'accent sur la solidité financière des contreparties, donne, en ce qui concerne les créances sur ou engagements envers des établissements de crédit, la priorité aux liens entre une succursale et sa maison-mère par rapport à la résidence de la succursale.

Exemple:
Un dépôt à vue effectué par le siège d'une banque japonaise établie à Tokyo auprès d'un établissement de crédit luxembourgeois est à renseigner sous pays «Reste du monde».
Par contre, un dépôt à vue effectué par une succursale d'une banque japonaise, établie dans la zone euro, est à renseigner sous pays «Autres EMUM».

Il y a lieu de procéder aux trois ventilations suivantes:

- Luxembourg (LU);
- Les autres pays membres de l'Union monétaire (Autres EMUM).
  Il s'agit des pays suivants:
  - Belgique
  - Allemagne,
    y compris Helgoland
  - France,
    y compris les départements d'Outre-mer (Guyane française, Guadeloupe, Martinique et Réunion), Saint Pierre et Miquelon, Mayotte et Monaco
  - Espagne,
    y compris les Iles Canaries et Ceuta et Melilla
  - Irlande
  - Italie
  - Pays-Bas
  - Autriche
  - Portugal,
    y compris les Açores et Madère
  - Finlande,
    y compris Ahvenanmaa
- Reste du monde: c'est-à-dire les pays qui ne tombent pas dans les deux catégories précitées.

4. Le secteur économique

Les montants sont à ventiler selon le secteur économique auquel appartient la contrepartie. Cette classification des secteurs économiques, qui est également utilisée pour l'établissement du tableau S 2.5 "Bilan statistique trimestriel", se présente comme suit.

4.1. Institutions financières monétaires (IFM)

Le secteur des institutions financières monétaires comprend toutes les sociétés et quasi-sociétés[2] financières exerçant, à titre principal, des activités d'intermédiation financière[3] consistant à recevoir des dépôts et/ou de proches substituts de dépôts de la part d'entités autres que des institutions financières monétaires, ainsi qu'à octroyer des crédits et/ou à effectuer des placements mobiliers pour leur compte propre.

La Banque centrale européenne met à la disposition des établissements déclarants une liste de toutes les institutions financières monétaires de l'Union européenne sur son site Internet (http://www.ecb.int) de façon à leur faciliter la tâche d'identifier correctement leurs contreparties. Cette liste commune est régulièrement mise à jour par les soins des banques centrales nationales.

Le secteur des institutions financières monétaires se subdivise en deux groupes d'institutions.

4.1.1. Etablissements de crédit

Relèvent notamment de ce groupe les intermédiaires financiers suivants:

- la Banque centrale européenne (BCE);
- les banques centrales nationales (BCN);
- les banques commerciales, les banques universelles et les banques à vocation polyvalente;
- les caisses d'épargne.

4.1.2. Autres institutions financières monétaires

Il s'agit des organismes de placement collectif tels que les fonds communs de placement, les sociétés d'investissement à capital variable, les sociétés d'investissement, etc. dans la mesure où ces intermédiaires financiers reçoivent des fonds du public, que ce soit sous la forme de dépôts ou de produits financiers qui sont des substituts proches des dépôts bancaires (p. ex. parts émises par des fonds d'investissement investissant dans des actifs très liquides, comme par exemple les instruments du marché monétaire.) Il y a lieu de reprendre dans cette catégorie uniquement les fonds d'investissement monétaires qui figurent sur la liste officielle des institutions financières monétaires que la Banque centrale européenne met à la disposition des établissements déclarants.

---

[2] Par quasi-société il faut entendre toute entité économique ayant une comptabilité propre mais étant dépourvue d'une personnalité juridique distincte.

[3] Selon le système européen des comptes nationaux SEC95, l'intermédiation financière est l'activité par laquelle une unité institutionnelle acquiert des actifs financiers et, simultanément, contracte des engagements pour son propre compte par le biais d'opérations financières sur le marché. Les actifs et passifs des intermédiaires financiers présentent des caractéristiques différentes, ce qui suppose que dans le cadre du processus d'intermédiation financière, les fonds collectés soient transformés ou regroupés sur la base de critères tels que l'échéance, le volume, le degré de risque, etc. (...) L'activité d'intermédiation financière consiste à mettre en présence une unité institutionnelle disposant de moyens excédentaires et une autre à la recherche de fonds. L'intermédiaire financier n'est pas simplement un agent agissant pour le compte de ces unités; il supporte lui-même un risque en acquérant des actifs financiers et en contractant des engagements pour son propre compte (SEC95, §2.32 -33 EUROSTAT juin 1996).

4.2. Administrations publiques

Le secteur public comprend:

- toutes les unités institutionnelles qui sont des autres producteurs non marchands[4] dont la production est destinée à la consommation individuelle et collective et dont la majeure partie des ressources provient de contributions obligatoires versées par des unités appartenant aux autres secteurs et/ou
- toutes les unités institutionnelles dont l'activité consiste à effectuer des opérations de redistribution du revenu et de la richesse nationaux.

Le secteur des administrations publiques se subdivise en deux sous-secteurs, à savoir l'administration publique centrale et les autres administrations publiques.

4.2.1. Administration publique centrale

Le secteur de l'administration publique centrale comprend tous les organismes centraux dont la compétence s'étend normalement sur la totalité du territoire économique, à l'exception des administrations de sécurité sociale de l'administration centrale.

Sont également à classer sous cette rubrique, le bureau des Comptes Chèques Postaux au Luxembourg ainsi que pour les autres pays membres de l'Union Monétaire les organismes de chèques et virements postaux qui ne sont pas des institutions financières monétaires mais qui ont une indépendance comptable. En ce qui concerne les pays classés dans la catégorie "Reste du monde", il y a lieu de reprendre sous cette rubrique les organismes de chèques et virements postaux qui ne sont pas des établissements de crédit mais qui ont une indépendance comptable.

4.2.2. Autres administrations publiques

Il y a lieu de regrouper ici l'ensemble des administrations publiques à l'exception de l'administration publique centrale.

- Administrations d'Etats fédérés
  Le secteur des administrations d'Etats fédérés réunit les administrations qui, en qualité d'unités institutionnelles distinctes, exercent certaines fonctions d'administration à un niveau inférieur à celui de l'administration centrale et supérieur à celui des unités publiques locales[5], à l'exception des administrations de sécurité sociale des administrations d'Etats fédérés.

- Administrations locales
  Le secteur des administrations locales rassemble toutes les administrations publiques dont la compétence s'étend seulement sur une subdivision locale du territoire économique, à l'exception des administrations de sécurité sociale des administrations locales.

[4] Dans la terminologie du SEC95, un autre producteur non marchand est un producteur dont la majeure partie de la production est cédée gratuitement ou à des prix économiquement non significatifs (SEC95, §3-23).

[5] De telles administrations sont par exemple les administrations des «Länder» allemands.

- Administrations de sécurité sociale
  Le secteur des administrations de sécurité sociale réunit toutes les unités institutionnelles centrales, fédérées et locales dont l'activité principale consiste à fournir des prestations sociales.

4.3. Autres secteurs

Cette catégorie regroupe l'ensemble des secteurs autres que les IFM et le secteur public. Il s'agit des secteurs suivants.

4.3.1. Autres intermédiations financières

Le secteur des autres intermédiaires financiers regroupe toutes les sociétés et quasi-sociétés financières dont la fonction principale consiste à fournir des services d'intermédiation financière en souscrivant des engagements sous des formes autres que du numéraire, des provisions techniques d'assurance ou des dépôts et/ou des proches substituts de dépôts provenant d'unités institutionnelles autres que des institutions financières monétaires.

Pour autant qu'elles ne soient pas des institutions financières monétaires, le secteur sous rubrique regroupe notamment les sociétés et quasi-sociétés financières suivantes:

- les holdings financiers;
- les organismes de placement collectif (OPC) tels que les fonds commun de placement (FCP), les sociétés à capital variable (SICAV), les sociétés d'investissement, etc., qui ne sont pas des «Institutions financières monétaires»
- les sociétés de crédit-bail;
- les sociétés exerçant des activités de location-vente, offrant des prêts personnels ou proposant des financements commerciaux;
- les sociétés d'affacturage;
- les courtiers en valeurs mobilières et produits financiers dérivés (travaillant pour leur compte propre);
- les sociétés financières spécialisées comme, par exemple, celles proposant du capital-risque, des capitaux d'amorçage ou des financements des exportations/importations;
- les sociétés-écrans créées pour détenir des actifs titrisés;
- les intermédiaires financiers qui reçoivent des dépôts et/ou des proches substituts de dépôts uniquement de la part d'institutions financières monétaires;
- les sociétés holding ayant pour objet unique de contrôler et de diriger un groupe de filiales dont l'activité principale consiste à fournir des services d'intermédiation financière et/ou à exercer des activités financières auxiliaires, mais qui ne sont pas elles-mêmes des sociétés financières.

4.3.2. Activités auxiliaires de l'intermédiation financière et activités auxiliaires de l'assurance

Le secteur des auxiliaires financiers comprend toutes les sociétés et quasi-sociétés financières dont la fonction principale consiste à exercer des activités financières auxiliaires, c'est-à-dire des activités étroitement liées à l'intermédiation financière ou à l'assurance mais n'en faisant pas partie.

Ce secteur comprend notamment:

- les courtiers d'assurance, les organismes de sauvetage et d'avarie, les conseillers en assurances et en pension; etc.;
- les courtiers de crédit, les courtiers en valeurs mobilières, les conseillers en placement, etc.;
- les sociétés d'émission de titres;
- les sociétés dont la fonction principale consiste à avaliser des effets et instruments analogues;
- les sociétés qui préparent (sans les émettre) des produits financiers dérivés et des instruments de couverture tels que des swaps, des options et des contrats à terme;
- les sociétés qui fournissent les infrastructures nécessaires au fonctionnement des marchés financiers;
- les autorités centrales de contrôle des intermédiaires financiers et des marchés financiers lorsqu'elles constituent des unités institutionnelles distinctes;
- les gestionnaires de fonds de pension, d'organismes de placement collectif, etc.;
- les bourses de valeurs mobilières ou les contrats d'assurance;
- les institutions sans but lucratif dotées de la personnalité juridique qui servent de sociétés financières, mais qui n'exercent aucune activité d'intermédiation financière ni aucune activité financière auxiliaire.

4.3.3. Sociétés d'assurances et fonds de pension

Il s'agit de toutes les sociétés et quasi-sociétés financières dont la fonction principale consiste à fournir des services d'intermédiation financière résultant de la mutualisation des risques.

Cette catégorie inclut notamment les fonds de pension sous forme de société d'épargne-pension à capital variable (sepcav) et d'association d'épargne-pension (assep) tels que définis par la loi du 8 juin 1999.

Sont à inclure également les sociétés d'assurances «captive» et de réassurances.

4.3.4. Sociétés et quasi-sociétés non financières du secteur public et privé

Le secteur des sociétés et quasi-sociétés non financières regroupe les unités institutionnelles dont les opérations de répartition et les opérations financières sont

distinctes de celles de leurs propriétaires et qui sont des producteurs marchands[6] dont la fonction principale consiste à produire des biens et des services non financiers.

Sont concernées les unités institutionnelles suivantes:

- les sociétés de capital privées et publiques qui sont des producteurs marchands dont la fonction principale consiste à produire des biens et des services non financiers;
- les sociétés coopératives et les sociétés de personnes dotées de la personnalité juridique qui sont des producteurs marchands dont la fonction principale consiste à produire des biens et des services non financiers;
- les producteurs publics dotés d'un statut qui leur confère la personnalité juridique qui sont des producteurs marchands dont la fonction principale consiste à produire des biens et des services non financiers;
- les institutions et associations sans but lucratif au service des sociétés non financières dotées de la personnalité juridique qui sont des producteurs marchands dont la fonction principale consiste à produire des biens et des services non financiers;
- les quasi-sociétés privées et publiques qui sont des producteurs marchands dont la fonction principale consiste à produire des biens et des services non financiers.

4.3.5. Ménages

Le secteur des ménages comprend les individus ou groupes d'individus tant dans leur fonction de consommateurs que dans celle, éventuelle, d'entrepreneurs produisant des biens marchands ou des services financiers et non financiers marchands, pour autant que, dans ce dernier cas, les activités correspondantes ne soient pas le fait d'unités distinctes traitées comme des quasi-sociétés. Ce secteur inclut également les individus ou groupes d'individus qui produisent des biens et des services non financiers exclusivement pour usage final propre.

Le secteur des ménages se subdivise en deux sous-secteurs.

4.3.5.1. Ménages - Personnes physiques

Le secteur des personnes physiques comprend:

- les individus ou groupes d'individus dont la fonction principale consiste à consommer;
- les individus ou groupes d'individus dont la fonction principale consiste à consommer et qui produisent des biens et des services non financiers exclusivement à usage final propre;
- les institutions sans but lucratif au service des ménages qui ne sont pas dotées de la personnalité juridique.

[6] Dans la terminologie du SEC95, on entend par production marchande la production écoulée ou destinée à être écoulée sur le marché.

Le secteur des personnes physiques comprend notamment:

- les salariés;
- les bénéficiaires de revenus de la propriété;
- les bénéficiaires d'autres revenus et de pensions.

4.3.5.2. Ménages - Entreprises individuelles

Le secteur des entreprises individuelles comprend les entreprises individuelles et les sociétés de personnes sans personnalité juridique (autres que des quasi-sociétés) qui sont des producteurs marchands.

4.3.6. Institutions sans but lucratif au service des ménages

Le secteur des institutions sans but lucratif au service des ménages regroupe les unités dotées de la personnalité juridique qui servent les ménages et qui sont des autres producteurs non marchands privés. Leurs ressources principales, autres que celles résultant des ventes occasionnelles, proviennent de contributions volontaires en espèces ou en nature effectuées par les ménages en leur qualité de consommateurs, de versements provenant des administrations publiques, ainsi que de revenus de la propriété.

5. Remarques

Un classement sectoriel et géographique particulier est applicable aux institutions internationales et supranationales.

Il y a lieu de distinguer entre:

5.1. Banque centrale européenne

Les ventilations suivantes sont applicables à la Banque centrale européenne:

Devise	A ventiler selon "Euro" ou "Autres devises"
Pays	Autres EMUM
Secteur économique	IFM: Etablissements de crédit

5.2. Institutions internationales et supranationales

Les ventilations suivantes sont applicables pour toutes les institutions internationales et supranationales indépendamment de leur type d'activité:

Devise	A ventiler selon "Euro" ou "Autres devises"
Pays	Reste du monde
Secteur économique	Administrations publiques

Annexe B

Situation active et passive

1.1. SITUATION ACTIVE ET PASSIVE : ACTIF

ETABLISSEMENT : DEXIA BANQUE INTERNATIONALE A LUXEMBOURG Identification2 Date de reporting : 2001-10-31 00:00:00 Date d'impression : 2001-12-17 Devises du capital : EUR		GARANTIES						GARANTIES		
		Montants bruts	Corrections de valeurs	Admin.centrales Banques centrales Zone A	Etablissements de crédit Zone A	Autres garanties réelles	Actifs liés	Hypothèques	Autres garanties personnelles	Prêts à la consommation
		(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)
		AMT	VAD	CGG	CIG	ORG	AAU	MGG	OPG	CDC
1-01.000	**Caisses, avoirs auprès des banques centrales et des offices des chèques postaux**									
1-01.100	Caisse									
1-01.200	Chèques postaux									
1-01.300	Banques centrales (à vue)									
1-01.310	*dont : Luxembourg/Belgique*									
1-01.320	*dont : autres pays zone A*									
1-02.000	**Effets publics et autres effets admissibles au refinancement auprès de la banque centrale**									
1-02.100	Effets publics et valeurs assimilées									
1-02.110	*dont : Luxembourg/Belgique*									
1-02.120	*dont : autres pays zone A*									
1-02.200	Autres effets admissibles au refinancement auprès de la banque centrale									
1-02.210	*dont : Luxembourg/Belgique*									
1-02.220	*dont : autres pays zone A*									
1-03.000	**Créances sur les établissements de crédit**									
1-03.100	**Créances sur les banques centrales**									
1-03.110	A vue									
1-03.111	Zone A (autre que Lux/Belgique)									
1-03.112	Zone B									
1-03.120	**Avec durée > à vue**									
1-03.121	Luxembourg/Belgique									
1-03.122	Zone A (autre)									
1-03.123	Zone B									
1-03.200	**Créances sur les banques multilatérales**									
1-03.210	A vue									
1-03.220	Avec durée > à vue									
1-03.300	**Créances sur les autres établissements de crédit**									
1-03.310	A vue									
1-03.311	Luxembourg									
1-03.312	Zone A (autres)									
1-03.313	Zone B									
1-03.320	Avec durée > à vue et<= 1 an									
1-03.321	Luxembourg									
1-03.322	Zone A(autres)									
1-03.323	Zone B									
1-03.330	**Avec durée > 1 an**									
1-03.331	Luxembourg									
1-03.332	Zone A (autres)									
1-03.333	Zone B									
1-03.340	*dont : sièges ou succursales*									
1-04.000	**Créances sur la clientèle**									
1-04.100	**Créances sur le secteur public**									
1-04.110	**Administrations centrales**									
1-04.111	Zone A									
1-04.112	Zone B									
1-04.120	**Administrations régionales et locales**									
1-04.121	Zone A									
1-04.122	Zone B									
1-04.200	**Créances sur la clientèle privée**									
1-04.210	**Personnes morales**									
1-04.211	Zone A									
1-04.212	Zone B									
1-04.220	**Personnes physiques**									
1-04.221	Zone A									
1-04.222	Zone B									

1.1. SITUATION ACTIVE ET PASSIVE : ACTIF

		GARANTIES						GARANTIES		
ETABLISSEMENT : DEXIA BANQUE INTERNATIONALE A Luxembourg Identification 2 Date de reporting : 2001-10-31 00:00:00 Date/Impression : 2001-12-17 Devises du capital : EUR		Montants bruts (1) AMT	Corrections de valeurs (2) VAD	Admin.centrales Banques centrales Zone A (3) CGG	Etablissements de crédit Zone A (4) CIG	Autres garanties réelles (5) ORG	Actifs liés (6) AAU	Hypothèques (7) MGG	Autres garanties personnelles (8) OPG	Prêts à la consommation (9) CDC
1-04.300	**Créances sur les établissements financiers**									
1-04.310	Zone A									
1-04.320	Zone B									
1-05.000	Opérations de crédit-bail									
1-05.100	Zone A									
1-05.200	Zone B									
1-06.000	**Obligations et autres valeurs immobilières à revenu fixe**									
1-06.100	Secteur Public									
1-06.110	Luxembourg									
1-06.120	Zone A (autres)									
1-06.130	Zone B									
1-06.200	Etablissements de crédit									
1-06.210	Luxembourg									
1-06.220	Zone A (autres)									
1-06.230	Zone B									
1-06.300	Autres émetteurs									
1-06.310	Luxembourg									
1-06.320	Zone A (autres)									
1-06.330	Zone B									
1-07.000	**Actions et autres valeurs mobilières à revevnu variable**									
1-07.100	Etablissements de crédit									
1-07.200	Etablissement financiers									
1-07.300	Autres									
1-08.000	Participations									
1-08.100	Etablissements de crédit									
1-08.200	Etablissements financiers									
1-08.300	Entreprises mises en équivalence									
1-08.400	Capital de dotation									
1-08.500	Autres									
1-09.000	**Parts dans des entreprises liées**									
1-09.100	Etablissements de crédit									
1-09.200	Etablissements financiers									
1-09.300	Entreprises mises en équivalence									
1-09.400	Autres									
1-10.000	**Actifs incorporels**									
1-10.100	Frais d'établissement									
1-10.200	Fonds de commerce acquis à titre onéreux									
1-10.300	Autres actifs incorporels									
1-80.000	**Différences de mise en equivalence par application de l'art.76**									
1-11.000	**Actifs corporels**									
1-11.100	Terrains et constructions									
1-11.110	*dont : utilisés dans le cadre de l'activité propre*									
1-11.200	Autres actifs corporels									
1-12.000	**Actions propres ou parts propres**									
1-13.000	**Autres actifs**									
1-13.100	Valeurs à recevoir à court terme									
1-13.200	Investissement du fonds de pension									
1-13.300	Primes d'options achetées									
1-13.400	Métaux précieux									
1-13.500	Autres									
1-14.000	Capital souscrit non versé									
1-14.100	*dont : appelé*									
1-15.000	Comptes de régularisation									
1	TOTAL ACTIF									

1.1. SITUATION ACTIVE ET PASSIVE : PASSIF

ETABLISSEMENT : DEXIA BANQUE INTERNATIONALE A LUXEMBOURG Identification 2 Date de reporting : 2001-10-31 00:00:00 Date d'impression : 2001-12-17 Devises du capital : EUR		MONTANTS AMT
2-01.000	Dettes envers les établissements de crédit	
2-01.100	Dettes envers les banques centrales	
2-01.110	A vue	
2-01.111	Luxembourg/Belgique	
2-01.112	Zone A (autres)	
2-01.113	Zone B	
2-01.120	**Avec durée > à vue**	
2-01.121	Luxembourg/Belgique	
2-01.122	Zone A (autres)	
2-01.123	Zone B	
2-01.200	**Dettes envers les banques multilatérales**	
2-01.210	A vue	
2-01.220	Avec durée > à vue	
2-01.300	**Dettes envers les autres établissements de crédit**	
2-01.310	A vue	
2-01.311	Luxembourg	
2-01.312	Zone A (autres)	
2-01.313	Zone B	
2-01.320	**Avec durée > à vue et <= 1 an**	
2-01.321	Luxembourg	
2-01.322	Zone A (autres)	
2-01.323	Zone B	
2-01.330	**Avec durée > 1 an**	
2-01.331	Luxembourg	
2-01.332	Zone A (autres)	
2-01.333	Zone B	

1.1. SITUATION ACTIVE ET PASSIVE : PASSIF

ETABLISSEMENT : DEXIA BANQUE INTERNATIONALE A LUXEMBOURG Identification 2 Date de reporting : 2001-10-31 00:00:00 Date d'impression : 2001-12-17 Devises du capital : EUR		MONTANTS AMT
2-01.340	*dont : établissements de crédit liés*	
2-01.341	*dont : sièges ou succursales*	
2-02.000	**Dettes envers la clientèle**	
2-02.100	**Dettes envers le secteur public**	
2-02.110	Dépôts d'épargne	
2-02.120	Comptes courants et autres dettes à vue	
2-02.130	Autres dettes à terme ou à préavis	
2-02.200	**Dettes envers la clientèle privée**	
2-02.210	**Personnes morales**	
2-02.211	Dépôts d'épargne	
2-02.212	Comptes courants et autres dettes à vue	
2-02.213	Autres dettes à terme ou à préavis	
2-02.220	**Personnes physiques**	
2-02.221	Dépôts d'épargne à vue	
2-02.222	Dépôts d'épargne à terme ou à préavis	
2-02.223	Comptes courants et autres dettes à vue	
2-02.224	Autres dettes à terme ou à préavis	
2-02.230	*dont: clients liés*	
2-02.240	*dont : établissements financiers*	
2-03.000	**Dettes représentées par un titre**	
2-03.100	Bons de caisse	
2-03.200	Obligations	
2-03.300	Titres du marché interbancaire et titres de créances négociables	
2-03.400	Autres dettes à terme ou à préavis	
2-04.000	**Autres passifs**	
2-04.100	Valeurs à payer à court terme	

1.1. SITUATION ACTIVE ET PASSIVE : PASSIF

ETABLISSEMENT : DEXIA BANQUE INTERNATIONALE A LUXEMBOURG Identification 2 Date de reporting : 2001-10-31 00:00:00 Date d'impression : 2001-12-17 Devises du capital : EUR		MONTANTS AMT
2-04.200	Créanciers Privilégiés	
2-04.300	Créanciers divers	
2-04.400	Fonds de pension en faveur du personnel	
2-04.500	Primes d'options vendues	
2-04.600	Autres	
2-05.000	**Comptes de régularisation**	
2-06.000	**Provisions pour risques et charges**	
2-06.100	Provisions pour pensions et obligations similaires	
2-06.200	Provision pour impôts	
2-06.300	**Autres provisions**	
2-06.310	sur passifs éventuels	
2-06.320	sur engagements hors-bilan	
2-06.330	sur opérations liées aux taux de change, aux taux d'intérêt et à d'autres cours de marché	
2-06.340	sur fonctions de gestion et de prise ferme	
2-06.350	Provisions AGDL	
2-06.360	Autres	
2-07.000	**Passifs subordonnés**	
2-07.100	Partie assimilée	
2-07.110	*dont : 'perpetuals' subordonnés*	
2-07.200	Partie non assimilée	
2-07.210	*dont : 'perpetuals' subordonnés*	
2-08.000	**Postes spéciaux avec une quote-part de réserves**	
2-08.100	**Plus-values de réinvestissement**	
2-08.110	Partie assimilée	
2-08.120	Partie non assimilée	

1.1. SITUATION ACTIVE ET PASSIVE : PASSIF

ETABLISSEMENT : DEXIA BANQUE INTERNATIONALE A LUXEMBOURG Identification 2 Date de reporting : 2001-10-31 00:00:00 Date d'impression : 2001-12-17 Devises du capital : EUR		MONTANTS AMT
2-08.200	**Plus-values neutralisées**	
2-08.210	Partie assimilée	
2-08.220	Partie non assimilée	
2-08.300	**Autres**	
2-08.310	Partie assimilée	
2-08.320	Partie non assimilée	
2-80.000	**Fonds pour risques bancaires généraux**	
2-81.000	**Corrections de valeur**	
2-81.100	Corrrections de valeur au sens de l'art. 62	
2-81.200	Provision forfaitaire	
2-81.300	Autres corrections de valeurs	
2-82.000	**Titres participatifs**	
2-09.000	**Capital souscrit (ou Capital de dotation)**	
2-09.100	*dont : actions privilégiées*	
2-10.000	**Primes d'émission**	
2-11.000	**Réserves**	
2-11.100	Réserve légale	
2-11.200	Réserve pour actions propres ou parts propres	
2-11.300	Réserves statuaires	
2-11.400	Autres réserves	
2-12.000	**Réserve de réévaluation**	
2-90.000	**Différences de mise en équivalence par application de l'art. 76**	
2-13.000	**Résultats reportés (+/-)**	
2-14.000	**Résultats de l'exercice (+/-)**	
2-15.000	**Résultat de l'exercice précédent en instance d'affectation (+/-)**	
2	**TOTAL PASSIF**	

Annexe C

Les fichiers en format XML

Regles.xml

Extrait

Nous présenterons ici un petit extrait des règles de regroupement telles qu'elles sont exprimées en format XML. Pour des raisons de présentation, les règles présentées sont telles qu'on peut les voir à l'aide d'un navigateur Internet et non pas telles qu'elles sont normalement codées en XML.

```
<fichier>
  <class id="C013" val="10120000">
  <ET>
    <ExprBool var="code_activite" op="=" arg="101" />
    <ExprBool var="pays_risk" op="=" arg="100#112#115#116#122#359" />
    <ExprBool var="duree_R" op="<=" arg="3" />
    <ExprBool var="W1i" op="=" arg="'103000E'" />
  </ET>
  </class>

  <class id="C013" val="10210000">
  <ET>
    <ExprBool var="W1i" op="=" arg="'102000E'#'204102E'" />
    <ExprBool var="espece_portf_ctv" op="<=" arg="0" />
  </ET>
  <ET>
    <ExprBool var="W1i" op="=" arg="'102000E'#'204102E'" />
    <ExprBool var="typ_rec_pos" op="=" arg="'2'#'3'#'5'#'6'#'7'" />
  </ET>
  </class>
</fichier>
```

Structure

Les marqueurs `<class> </class>` délimitent la définition d'une règle pour la classification spécifiée à l'aide des attributs.
Les marqueurs `<ET> </ET>` délimitent la définition d'une sous-règle, c'est-à-dire une conjonction d'expressions booléennes.
Les marqueurs `<ExprBool> </ExprBool>` délimitent la définition d'une expression booléenne.
Les marqueurs `<compl> </compl>` peuvent être utilisé pour délimiter la définition d'un complémentaire.

Requetes.xml

Extrait

```
<fichier>
<variable
        id='code_activite'
        type='int'
        tag='1'
        req='SELECT distinct(Code_Activite_Comptable)
             FROM DETAIL_BILANTAIRE'
        dom=''>
</variable>

<variable
        id='duree_R'
        type='int'
        tag='0'
        req=''
```

```
        dom='0..3500'>
</variable>

<variable
        id='typ_rec_pos'
        type='char'
        tag='0'
        req=''
        dom='1,2,3,4,5,6,7'>
</variable>

<variable
        id='type_rap'
        type='smallint'
        tag='1'
        req='SELECT distinct(TYPE_RAPPORT_N003.Type_Rapport)
             FROM TYPE_RAPPORT_N003'
        dom=''>
</variable>

</fichier>
```

Structure

Pour chaque variable du problème on définira un marqueur `<variable> </variable>` avec cinq attributs. Ces cinq attributs sont les suivants :

- `id` : identifiant de la variable ;
- `type` : type de la variable (char, varchar, int, ...) ;
- `tag` : 0 ou 1

  0 indique qu'on utilise le domaine défini avec l'attribut `dom` lors de la création

automatique du fichier `Domaines.xml`

1 indique qu'on demande l'exécution de la requête définie avec l'attribut `req` lors de la création automatique du fichier `Domaines.xml` ;

- `req` : requête SQL pour récupérer le domaine dans la base de données ;
- `dom` : domaine de la variable, défini 'manuellement'.

Domaines.xml

Extrait

```
<fichier>
<variable
    id='CODE_ACTIVITE'
    dom='999, 101, 110, 111, 112, 114, 120, 121, 122, 125, 130, 131, 132,
         133, 135, 136, 140, 141, 142, 150, 151, 152, 153, 220, 221, 222,
         223, 224, 225, 226, 227, 230, 231, 232, 233, 234, 239, 241, 242,
         243, 244, 262, 263, 267, 300, 501, 502, 503, 504, 505'>
</variable>

<variable
    id='DUREE_R'
    dom='0..3500'>
</variable>

<variable
    id='TYP_REC_POS'
    dom=''1','2','3','4','5','6','7''>
</variable>
```

```
<variable
    id='TYPE_RAP'
    dom='0 , 1 , 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23,
         24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 40, 50, 55, 60,
         61, 62, 64, 66, 80, 82, 84, 85, 90, 95'>
</variable>

</fichier>
```

Structure

Pour chaque variable, on définira un marqueur `<variable></variable>` possédant deux attributs. Le premier attribut (`id`) sera l'identifiant de la variable et le second attribut (`dom`) sera le domaine de la variable.

Remarque : Les valeurs du domaines peuvent être de type caractères. Ces valeurs sont entourées par des apostrophes ('). Les apostrophes doivent être remplacées par un caractère d'échappement en XML qui est &apos ;. exemple : '100200E' devra être écrit en XML &apos ;100200E&apos ;

Annexe D

Résultats obtenus

Compatibilité

Règle testée

```
SI code_activite = 141#142
∧ duree_R <= 3
∧ pays_risk = 100#101#111#112#113#114#121#122#123#124#125#126#127#141#
              143#145#147#149#151#156#157#311#387#401#403#405#501#509
∧ duree_R > 31
∧ W1i = '107000E'
ALORS C013=10311100
```

Règle testée en XML

```
<class id="C013" val="10311100">
<ET>
  <ExprBool var="code_activite" op="=" arg="141#142" />
  <ExprBool var="duree_R" op="<=" arg="3" />
  <ExprBool var="pays_risk" op="=" arg="100#101#111#112#113#114#121#122#
                                        123#124#125#126#127#141#143#145#
                                        147#149#151#156#157#311#387#401#
                                        403#405#501#509" />
  <ExprBool var="duree_R" op=">" arg="31" />
  <ExprBool var="W1i" op="=" arg="'107000E'" />
</ET>
</class>
```

Code ECL^iPS^e

```
:-lib(fd).

final(V):-
V=[CODE_ACTIVITE, DUREE_R, PAYS_RISK, W1I],
CODE_ACTIVITE::[ 999, 101, 110, 111, 112, 114, 120, 121, 122, 125,
                 130, 131, 132, 133, 135, 136, 140, 141, 142, 150,
                 151, 152, 153, 220, 221, 222, 223, 224, 225, 226,
                 227, 230, 231, 232, 233, 234, 239, 241, 242, 243,
                 244, 262, 263, 267, 300, 501, 502, 503, 504, 505],
PAYS_RISK::[  0, 100, 101, 111, 112, 113, 114, 115, 116, 117, 118,
            119, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130,
            131, 141, 143, 145, 147, 149, 151, 153, 154, 155, 156,
            157, 158, 159, 161, 163, 165, 166, 167, 168, 169, 171,
            173, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190,
            191, 192, 193, 194, 195, 196, 199, 201, 202, 203, 205,
            207, 209, 211, 213, 215, 219, 221, 223, 225, 227, 229,
            231, 233, 235, 237, 239, 241, 243, 245, 247, 249, 250,
            251, 253, 255, 257, 259, 260, 261, 263, 265, 267, 269,
            271, 272, 273, 275, 277, 278, 279, 280, 281, 282, 283,
            285, 286, 287, 289, 291, 293, 295, 296, 297, 298, 301,
            303, 305, 307, 311, 315, 321, 322, 324, 325, 327, 329,
            331, 333, 335, 337, 339, 343, 344, 345, 346, 347, 351,
            353, 355, 357, 359, 363, 365, 368, 371, 373, 374, 375,
            377, 384, 385, 386, 387, 388, 389, 391, 401, 403, 405,
            407, 409, 411, 413, 415, 417, 419, 421, 422, 423, 424,
            425, 427, 429, 431, 433, 436, 438, 439, 440, 441, 442,
            443, 445, 447, 449, 455, 457, 459, 461, 463, 465, 471,
            473, 475, 479, 481, 483, 485, 488, 501, 503, 505, 509,
```

```
            510, 511, 512, 513, 520, 524, 525, 541, 543, 544, 545,
            546, 547, 548, 549, 551, 553, 554, 555, 557, 559, 560,
            561, 562, 563, 564, 565, 567, 568, 569, 570, 571, 572,
            573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583,
            600, 605, 699, 808, 912, 920, 921, 984],
DUREE_R::[0..3500],
W1I::[ '204113E','207000 ','3031127','3031128','3031307','3031308',
       '3031522','204109E','109000E','204108E','108000E','107000E',
        '204107E','103000E','104000E','204102E','102000E','3042002',
        '3042003','106000E','204106E','112000E','113100E','3042001',
        '3042002','3042003','3042004','204112E','105000E','201000E',
        '202000E'],

(CODE_ACTIVITE#=141 #\/ CODE_ACTIVITE#=142) #/\
(DUREE_R#<=3) #/\
(PAYS_RISK#=100 #\/ PAYS_RISK#=101 #\/ PAYS_RISK#=111 #\/
 PAYS_RISK#=112 #\/ PAYS_RISK#=113 #\/ PAYS_RISK#=114 #\/
 PAYS_RISK#=121 #\/ PAYS_RISK#=122 #\/ PAYS_RISK#=123 #\/
 PAYS_RISK#=124 #\/ PAYS_RISK#=125 #\/ PAYS_RISK#=126 #\/
 PAYS_RISK#=127 #\/ PAYS_RISK#=141 #\/ PAYS_RISK#=143 #\/
 PAYS_RISK#=145 #\/ PAYS_RISK#=147 #\/ PAYS_RISK#=149 #\/
 PAYS_RISK#=151 #\/ PAYS_RISK#=156 #\/ PAYS_RISK#=157 #\/
 PAYS_RISK#=311 #\/ PAYS_RISK#=387 #\/ PAYS_RISK#=401 #\/
 PAYS_RISK#=403 #\/ PAYS_RISK#=405 #\/ PAYS_RISK#=501 #\/
 PAYS_RISK#=509) #/\
(DUREE_R#>31) #/\
(W1I#='107000E'),
once labeling(V).
```

L'évaluation de cette sous-règle par ECL^iPS^e se soldera par un échec. On sait

alors qu'il existe une incompatibilité dans cette règle mais il faut en déterminer la cause. Les expressions booléennes définies sur une même variables seront alors testées deux à deux, ce qui mènera au code suivant :

```
:-lib(fd).

final(V):-
    V=[DUREE_R],
    DUREE_R::[0..3500],
    ( (DUREE_R#<=3)#/\(DUREE_R#>31) ),
    once labeling(V).
```

Résultats obtenus

```
L'expression (DUREE_R#<=3)#/\(DUREE_R#>31) est incompatible.

Chaque élément de l'expression est compatible avec son domaine.
```

Recouvrement

Règles testées

```
SI code_activite = 101#140
∧ pays_risk = 100#112#115#116#122#359
∧ duree_R <= 3
∧ W1i = '105000E'
ALORS C013=10120000

SI code_activite = 101#102#103#104#105#106#107#108#109#110#111#
                   112#113#114#115#116#117#118#119#120#121#122#
                   123#124#125#126#127#128#129#130#131#132#133#
                   134#135#136#137#138#139#153
∧ duree_R <= 3
∧ pays_risk = 116
∧ W1i = '104000E'
ALORS C013 = 10331100
```

Règles testées en XML

```
class id="C013" val="10120000">
<ET>
  <ExprBool var="code_activite" op="=" arg="101#140" />
  <ExprBool var="pays_risk" op="=" arg="100#112#115#116#122#359" />
  <ExprBool var="duree_R" op="<=" arg="3" />
  <ExprBool var="W1i" op="=" arg="'105000E'" />
</ET>
</class>

<class id="C013" val="10331100">
```

```
<ET>
  <ExprBool var="code_activite" op="=" arg="101#102#103#104#105#106#107#
                                            108#109#110#111#112#113#114#
                                            115#116#117#118#119#120#121#
                                            122#123#124#125#126#127#128#
                                            129#130#131#132#133#134#135#
                                            136#137#138#139#153" />
  <ExprBool var="duree_R" op="<=" arg="3" />
  <ExprBool var="pays_risk" op="=" arg="116" />
  <ExprBool var="W1i" op="=" arg="'104000E'" />
</ET>
 </class>
```

Code ECL^iPS^e

Remarque : Dans la suite, les domaines ne seront plus énumérés en intégralité comme dans l'exemple donné dans le cas de l'incompatibilité.

```
:-lib(fd).

final(V):- V=[W1I, CODE_ACTIVITE, PAYS_RISK, DUREE_R],
CODE_ACTIVITE::[ 101, ..., 505],
PAYS_RISK::[  0, ... , 984],
DUREE_R::[0..3500],
W1I::[ '204113E', ..., '202000E'],
( (CODE_ACTIVITE#=101) #/\
  ( PAYS_RISK#=100 #\/ PAYS_RISK#=112 #\/ PAYS_RISK#=115 #\/
    PAYS_RISK#=116 #\/ PAYS_RISK#=122 #\/ PAYS_RISK#=359
  ) #/\
  (DUREE_R#<=3) #/\  (W1I#='103000E')
)
```

```
#/\
( (CODE_ACTIVITE#=101 #\/ CODE_ACTIVITE#=102 #\/ CODE_ACTIVITE#=103 #\/
   CODE_ACTIVITE#=104 #\/ CODE_ACTIVITE#=105 #\/ CODE_ACTIVITE#=106 #\/
   CODE_ACTIVITE#=107 #\/ CODE_ACTIVITE#=108 #\/ CODE_ACTIVITE#=109 #\/
   CODE_ACTIVITE#=110 #\/ CODE_ACTIVITE#=111 #\/ CODE_ACTIVITE#=112 #\/
   CODE_ACTIVITE#=113 #\/ CODE_ACTIVITE#=114 #\/ CODE_ACTIVITE#=115 #\/
   CODE_ACTIVITE#=116 #\/ CODE_ACTIVITE#=117 #\/ CODE_ACTIVITE#=118 #\/
   CODE_ACTIVITE#=119 #\/ CODE_ACTIVITE#=120 #\/ CODE_ACTIVITE#=121 #\/
   CODE_ACTIVITE#=122 #\/ CODE_ACTIVITE#=123 #\/ CODE_ACTIVITE#=124 #\/
   CODE_ACTIVITE#=125 #\/ CODE_ACTIVITE#=126 #\/ CODE_ACTIVITE#=127 #\/
   CODE_ACTIVITE#=128 #\/ CODE_ACTIVITE#=129 #\/ CODE_ACTIVITE#=130 #\/
   CODE_ACTIVITE#=131 #\/ CODE_ACTIVITE#=132 #\/ CODE_ACTIVITE#=133 #\/
   CODE_ACTIVITE#=134 #\/ CODE_ACTIVITE#=135 #\/ CODE_ACTIVITE#=136 #\/
   CODE_ACTIVITE#=137 #\/ CODE_ACTIVITE#=138 #\/ CODE_ACTIVITE#=139 #\/
   CODE_ACTIVITE#=153)#/\
  (DUREE_R#<=3)#/\
  (PAYS_RISK#=116)#/\
  (W1I#='103000E' #\/ W1I#='104000E')
),
once labeling(V).
```

Résultats obtenus

```
Il y a recouvrement entre les 2 règles pour les valeurs
[W1I, CODE_ACTIVITE, PAYS_RISK, DUREE_R]=[103000E, 101, 116, 0]

Ce sont les règles de valeur 10120000
                          et 10331100

Ces deux règles sont les suivantes :
```

```
[ [CODE_ACTIVITE#=101, PAYS_RISK#=100 #\/ PAYS_RISK#=112 #\/
   PAYS_RISK#=115 #\/  PAYS_RISK#=116 #\/ PAYS_RISK#=122 #\/
   PAYS_RISK#=359, DUREE_R#<=3, W1I#='103000E'] ]

[ [CODE_ACTIVITE#=101 #\/ CODE_ACTIVITE#=102 #\/ CODE_ACTIVITE#=103 #\/
   CODE_ACTIVITE#=104 #\/ CODE_ACTIVITE#=105 #\/ CODE_ACTIVITE#=106 #\/
   CODE_ACTIVITE#=107 #\/ CODE_ACTIVITE#=108 #\/ CODE_ACTIVITE#=109 #\/
   CODE_ACTIVITE#=110 #\/ CODE_ACTIVITE#=111 #\/ CODE_ACTIVITE#=112 #\/
   CODE_ACTIVITE#=113 #\/ CODE_ACTIVITE#=114 #\/ CODE_ACTIVITE#=115 #\/
   CODE_ACTIVITE#=116 #\/ CODE_ACTIVITE#=117 #\/ CODE_ACTIVITE#=118 #\/
   CODE_ACTIVITE#=119 #\/ CODE_ACTIVITE#=120 #\/ CODE_ACTIVITE#=121 #\/
   CODE_ACTIVITE#=122 #\/ CODE_ACTIVITE#=123 #\/ CODE_ACTIVITE#=124 #\/
   CODE_ACTIVITE#=125 #\/ CODE_ACTIVITE#=126 #\/ CODE_ACTIVITE#=127 #\/
   CODE_ACTIVITE#=128 #\/ CODE_ACTIVITE#=129 #\/ CODE_ACTIVITE#=130 #\/
   CODE_ACTIVITE#=131 #\/ CODE_ACTIVITE#=132 #\/ CODE_ACTIVITE#=133 #\/
   CODE_ACTIVITE#=134 #\/ CODE_ACTIVITE#=135 #\/ CODE_ACTIVITE#=136 #\/
   CODE_ACTIVITE#=137 #\/ CODE_ACTIVITE#=138 #\/ CODE_ACTIVITE#=139 #\/
   CODE_ACTIVITE#=153,
   DUREE_R#<=3, PAYS_RISK#=116, W1I#='103000E' #\/ W1I#='104000E'] ]

Le recouvrement est du aux 2 sous-règles suivantes :

Sous-règle 1 de la règle 10120000 :
[CODE_ACTIVITE#=101, PAYS_RISK#=100 #\/ PAYS_RISK#=112 #\/
 PAYS_RISK#=115 #\/  PAYS_RISK#=116 #\/ PAYS_RISK#=122 #\/
 PAYS_RISK#=359, DUREE_R#<=3, W1I#='103000E']

Sous-règle 1 de la règle 10331100 :
[CODE_ACTIVITE#=101 #\/ CODE_ACTIVITE#=102 #\/ CODE_ACTIVITE#=103 #\/
```

```
CODE_ACTIVITE#=104 #\/ CODE_ACTIVITE#=105 #\/ CODE_ACTIVITE#=106 #\/
CODE_ACTIVITE#=107 #\/ CODE_ACTIVITE#=108 #\/ CODE_ACTIVITE#=109 #\/
CODE_ACTIVITE#=110 #\/ CODE_ACTIVITE#=111 #\/ CODE_ACTIVITE#=112 #\/
CODE_ACTIVITE#=113 #\/ CODE_ACTIVITE#=114 #\/ CODE_ACTIVITE#=115 #\/
CODE_ACTIVITE#=116 #\/ CODE_ACTIVITE#=117 #\/ CODE_ACTIVITE#=118 #\/
CODE_ACTIVITE#=119 #\/ CODE_ACTIVITE#=120 #\/ CODE_ACTIVITE#=121 #\/
CODE_ACTIVITE#=122 #\/ CODE_ACTIVITE#=123 #\/ CODE_ACTIVITE#=124 #\/
CODE_ACTIVITE#=125 #\/ CODE_ACTIVITE#=126 #\/ CODE_ACTIVITE#=127 #\/
CODE_ACTIVITE#=128 #\/ CODE_ACTIVITE#=129 #\/ CODE_ACTIVITE#=130 #\/
CODE_ACTIVITE#=131 #\/ CODE_ACTIVITE#=132 #\/ CODE_ACTIVITE#=133 #\/
CODE_ACTIVITE#=134 #\/ CODE_ACTIVITE#=135 #\/ CODE_ACTIVITE#=136 #\/
CODE_ACTIVITE#=137 #\/ CODE_ACTIVITE#=138 #\/ CODE_ACTIVITE#=139 #\/
CODE_ACTIVITE#=153,
DUREE_R#<=3, PAYS_RISK#=116, W1I#='103000E' #\/ W1I#='104000E']
```

Remarque : la virgule correspond au connecteur $\wedge$. Nous avons ici un cas particulier où les deux règles ne sont composées que d'une seule sous-règle. C'est pourquoi le descriptif de chacune des règles est identique à celui de la sous-règle.

Définition complète

Afin de limiter la longueur de la description des règles ainsi que du code ECL^iPS^e à introduire dans cette annexe, nous donnerons ici un exemple de test effectué sur une classification composée de seulement deux règles.

Règles testées

```
SI code_activite = 101#140
∧   pays_risk = 100#112#115#116#122#359
∧   duree_R <= 3
∧   W1i = '105000E'
ALORS C013=10120000

SI code_activite = 140#141#142
∧ duree_R <= 3
∧   W1i = '105000E'
ALORS C013=10130000
```

Règles testées en XML

```
<class id="C013" val="10120000">
<ET>
  <ExprBool var="code_activite" op="=" arg="101#140" />
  <ExprBool var="pays_risk" op="=" arg="100#112#115#116#122#359" />
  <ExprBool var="duree_R" op="<=" arg="3" />
  <ExprBool var="W1i" op="=" arg="'105000E'" />
</ET>
</class>
```

```
<class id="C013" val="10130000">
<ET>
  <ExprBool var="code_activite" op="=" arg="140#141#142" />
  <ExprBool var="duree_R" op="<=" arg="3" />
  <ExprBool var="W1i" op="=" arg="'105000E'" />
</ET>
</class>
```

Code ECL^iPS^e

```
:-lib(fd).

final(V):-
V=[CODE_ACTIVITE, PAYS_RISK, DUREE_R, W1I],
CODE_ACTIVITE::[ 101, ..., 505],
PAYS_RISK::[  0, ..., 984],
DUREE_R::[0..3500],
W1I::[ '204113E', ..., '202000E'],
#\+( (
      ( (CODE_ACTIVITE#=101 #\/ CODE_ACTIVITE#=140)
        #/\  (PAYS_RISK#=100 #\/ PAYS_RISK#=112 #\/ PAYS_RISK#=115 #\/
              PAYS_RISK#=116 #\/ PAYS_RISK#=122 #\/ PAYS_RISK#=359)
        #/\  (DUREE_R#<=3)   #/\ (W1I#='105000E')
      )
      #\/
      ( (CODE_ACTIVITE#=140 #\/ CODE_ACTIVITE#=141 #\/ CODE_ACTIVITE#=142)
       #/\  (DUREE_R#<=3)   #/\ (W1I#='105000E')
      )
    )
  ), once labeling(V).
```

Résultats obtenus

La classification C013 n'est pas définie pour
[CODE_ACTIVITE, PAYS_RISK, DUREE_R, W1I] = [101, 0, 0, 102000E]