

Université de Montréal

**Un modèle uniforme pour la modélisation  
et la métamodélisation d'une mémoire  
d'entreprise**

par

**Olivier Gerbé**

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

Thèse présentée à la Faculté des études supérieures  
en vue de l'obtention du grade de  
Philosophiae Doctor (Ph.D.)  
en informatique

janvier, 2000

© Olivier Gerbé, 2000

Université de Montréal  
Faculté des études supérieures

Cette thèse intitulée:

Un modèle uniforme pour la modélisation  
et la métamodélisation d'une mémoire  
d'entreprise

présentée par

Olivier Gerbé

a été évaluée par un jury composé des personnes suivantes:

---

Esma Aïmeur  
(présidente du jury)

---

Houari Sahraoui  
(membre du jury)

---

Rudolf Keller  
(directeur de recherche)

---

Guy Mineau  
(codirecteur de recherche)

---

John Sowa  
(examineur externe)

---

Samuel Pierre  
(représentant du doyen)

Thèse acceptée le 5 avril 2000

# Sommaire

De nos jours, les entreprises manipulent de gros volumes d'information toujours plus complexes, aussi bien dans leur nature que dans leur structure. Ces informations peuvent être regroupées en deux catégories : les données opérationnelles qui sont utilisées pour la gestion des entreprises et les connaissances corporatives qui constituent le savoir-faire de l'entreprise.

Cette dernière catégorie fait actuellement l'objet d'importants travaux de recherche portant sur la représentation des savoir-faire de l'entreprise. Il s'agit alors de représenter d'une part les connaissances formelles comme les processus d'affaires, les procédures ou les politiques (mission, règlements, normes) et d'autre part les connaissances informelles comme les meilleures pratiques ou encore la culture de l'entreprise.

Dans les systèmes de gestion traditionnels, les données opérationnelles sont structurées conformément à des modèles classiques (système comptable, système de paie, etc.) et évoluent peu. Par opposition, les connaissances corporatives sont souvent non structurées ou semi-structurées, elles évoluent beaucoup et elles nécessitent la description de la connaissance sur ces données (métadonnées). Dans ce contexte, les systèmes de gestion de bases de données existants (relationnels ou orientés objet) sont peu adaptés car les traitements des données et des métadonnées diffèrent.

Dans cette thèse, nous proposons une approche basée sur le traitement uniforme des données et métadonnées. Nos travaux portent sur la spécification d'un formalisme capable de représenter d'une manière uniforme ces deux types de données. Ces travaux de métamodélisation sont basés sur les graphes conceptuels, un formalisme puissant, fondé sur la logique et permettant de modéliser les objets du discours par des concepts et des relations conceptuelles. À partir de ce formalisme, nous proposons un langage et un ensemble d'opérateurs permettant de modéliser et de manipuler d'une manière uniforme les différents niveaux de représentation de la connaissance.

Ce formalisme et ce langage constituent un premier pas vers la réalisation d'outils

permettant aux entreprises de capitaliser sur leur savoir-faire. Cette approche a été validée d'une manière théorique à travers la modélisation des modèles d'expertise de la méthode CommonKADS. Enfin une validation pratique a été réalisée par la constitution de mémoires d'entreprise contenant le savoir-faire et les meilleurs pratiques dans des domaines tels que l'aéronautique, les technologies de l'information et les banques.

# Table des matières

<b>Liste des Figures</b>	<b>iv</b>
<b>Liste des Tables</b>	<b>x</b>
<b>Introduction</b>	<b>1</b>
<b>Chapitre 1 : Contexte</b>	<b>8</b>
1.1 La notion d'information . . . . .	9
1.2 L'informatisation des entreprises . . . . .	11
1.3 La mémoire d'entreprise . . . . .	13
<b>Chapitre 2 : État de l'art</b>	<b>16</b>
2.1 Les types de connaissances . . . . .	16
2.2 Les niveaux de modélisation . . . . .	17
2.2.1 Les données : le projet . . . . .	18
2.2.2 Le modèle : la méthode . . . . .	18
2.2.3 Le métamodèle : le formalisme . . . . .	19
2.2.4 Le méta-métamodèle : le modèle de connaissance . . . . .	20
2.3 Formalismes de structure . . . . .	20
2.3.1 Notions de base . . . . .	20
2.3.2 Les besoins . . . . .	21
2.3.3 Modèle Entité-Relation Étendu . . . . .	26
2.3.4 Formalisme Orienté Objet . . . . .	27
2.3.5 Le modèle relationnel . . . . .	28
2.3.6 Classic . . . . .	30
2.3.7 Graphes conceptuels . . . . .	32

2.3.8	Synthèse . . . . .	33
2.4	Formalismes de dynamique . . . . .	33
2.4.1	Notions de base . . . . .	34
2.4.2	Les besoins . . . . .	36
2.4.3	Unified Modeling Language . . . . .	37
2.4.4	Process Interchange Format (PIF) . . . . .	39
2.4.5	Workflow Reference Model . . . . .	43
2.4.6	Graphes conceptuels et processus . . . . .	46
2.4.7	Synthèse . . . . .	49
2.5	Conclusion . . . . .	49
<b>Chapitre 3 : Le langage</b>		<b>50</b>
3.1	Le langage et la pyramide des modèles . . . . .	50
3.1.1	Notation et description du modèle uniforme . . . . .	52
3.1.2	Exemples . . . . .	53
3.1.3	Graphes Conceptuels et métamodélisation . . . . .	55
3.2	Le langage . . . . .	58
3.2.1	Principes . . . . .	58
3.2.2	Opérations de base sur les graphes conceptuels . . . . .	59
3.2.3	Vue d'ensemble du langage . . . . .	61
3.2.4	Les types de concepts du langage . . . . .	64
3.2.5	Les relations conceptuelles du langage . . . . .	80
3.3	Opérateur de correspondance . . . . .	86
3.4	Contexte . . . . .	89
3.5	Sommaire . . . . .	95
<b>Chapitre 4 : Méta-métamodèle et métamodèles</b>		<b>97</b>
4.1	Méta-métamodèle . . . . .	99
4.1.1	Introduction . . . . .	99
4.1.2	Éléments du méta-métamodèle . . . . .	100
4.2	Métamodèles . . . . .	102
4.2.1	Introduction . . . . .	102
4.2.2	Noyau commun . . . . .	103

4.2.3	Métamodèle de structure . . . . .	104
4.2.4	Métamodèle de dynamique . . . . .	106
4.2.5	Métamodèle de recherche . . . . .	109
4.2.6	Métamodèle de contrainte . . . . .	117
4.3	Un exemple . . . . .	117
4.4	Sommaire . . . . .	121
<b>Chapitre 5 : Évaluation du modèle uniforme</b>		<b>124</b>
5.1	Évaluation théorique . . . . .	124
5.1.1	Besoins concernant la partie statique de la connaissance . . . .	125
5.1.2	Besoins concernant la partie dynamique de la connaissance . .	130
5.1.3	Besoins concernant les contraintes . . . . .	132
5.1.4	Besoins concernant la recherche . . . . .	141
5.1.5	Confrontation avec CommonKads . . . . .	142
5.1.6	Évaluation théorique : Sommaire . . . . .	154
5.2	Évaluation pratique . . . . .	154
5.2.1	Ontologie : Noyau . . . . .	155
5.2.2	Ontologie : Système . . . . .	160
5.2.3	Conclusion de l'évaluation pratique . . . . .	174
5.3	Sommaire . . . . .	176
<b>Conclusion</b>		<b>178</b>
<b>Références</b>		<b>182</b>
<b>Annexe A : Introduction aux graphes conceptuels</b>		<b>190</b>

# Liste des Figures

2.1	Architecture à quatre niveaux. . . . .	18
2.2	La méthode. . . . .	19
2.3	Le formalisme. . . . .	19
2.4	Un méta-métamodèle simplifié. . . . .	20
2.5	Un exemple de modèle avec deux catégories et une relation. . . . .	21
2.6	Différentes hiérarchies basées sur trois critères. . . . .	22
2.7	Une hiérarchie basée sur les trois critères. . . . .	23
2.8	Le modèle complet. . . . .	24
2.9	Catégorie ou instance. . . . .	25
2.10	Entité-Relation Étendu - Tout employé travaille pour une organisation. . . . .	26
2.11	Entité-Relation Étendu - Relation entre catégories et/ou instance. . . . .	27
2.12	Entité-Relation Étendu - Catégorie ou instance. . . . .	27
2.13	UML - Tout employé travaille pour une organisation. . . . .	28
2.14	UML - Relation entre catégories et/ou instances. . . . .	28
2.15	UML - Catégorie ou instance. . . . .	29
2.16	Modèle relationnel - Tables et attributs de jointure. . . . .	29
2.17	Modèle relationnel - Relations entre catégories et/ou instances. . . . .	30
2.18	Modèle relationnel - Catégorie ou instance. . . . .	30
2.19	Classic - Concepts et relations. . . . .	31
2.20	Classic - Relation entre catégories et/ou instances. . . . .	31
2.21	Graphes conceptuels - Relation entre catégories et/ou instances. . . . .	32
2.22	Graphes conceptuels : Catégorie ou instance. . . . .	33
2.23	Une activité avec ses participants. . . . .	35
2.24	Un processus comme une suite d'activités. . . . .	35
2.25	Le partage d'activités. . . . .	36



2.26	Le problème de la fenêtre. . . . .	37
2.27	UML - Représentation d'une activité et d'un processus. . . . .	39
2.28	UML - Représentation du problème de la fenêtre. . . . .	39
2.29	Le métamodèle de PIF. . . . .	40
2.30	Le métamodèle du WfMC. . . . .	44
3.1	Pyramide des modèles. . . . .	51
3.2	Définition du type Conducteur. . . . .	52
3.3	Exemple de graphe conceptuel du niveau données. . . . .	53
3.4	Exemple de graphe conceptuel du niveau modèle. . . . .	54
3.5	Exemple de graphe conceptuel du niveau métamodèle. . . . .	55
3.6	Exemple de graphe conceptuel du niveau métamétamodèle. . . . .	55
3.7	Exemple de graphe conceptuel du langage. . . . .	56
3.8	Hierarchie des types de concepts du langage. . . . .	61
3.9	Le modèle du langage - partie 1(formalisme UML). . . . .	62
3.10	Le modèle du langage - partie 2 (formalisme UML). . . . .	63
3.11	Hierarchie des types de relations du langage. . . . .	63
3.12	Graphe de spécification de Graph. . . . .	64
3.13	Graphe de spécification de Concept. . . . .	65
3.14	Graphe de spécification de Referent. . . . .	66
3.15	Graphe de spécification de GenericConcept. . . . .	67
3.16	Graphe de spécification de IndividualConcept. . . . .	68
3.17	Graphe de spécification de CoRefLink. . . . .	69
3.18	Graphe de spécification de DefiningConcept. . . . .	70
3.19	Graphe de spécification de BoundConcept. . . . .	70
3.20	Graphe de spécification de Arc. . . . .	71
3.21	Source et destination. . . . .	71
3.22	Graphe de spécification de SrceArc. . . . .	72
3.23	Graphe de spécification de ArcDest. . . . .	72
3.24	Graphe de spécification de Relation. . . . .	73
3.25	Graphe de spécification de Type. . . . .	74
3.26	Graphe de spécification de ConceptType. . . . .	75
3.27	Graphe de la spécification de Québécois. . . . .	75

3.28	Graphe simplifié de la spécification de Québécois. . . . .	76
3.29	Graphe de spécification de RelationType. . . . .	76
3.30	Graphe de la spécification de la relation GoingTo. . . . .	76
3.31	Graphe de la spécification de la relation rsubt. . . . .	77
3.32	Graphe simplifié de la spécification de rsubt. . . . .	77
3.33	Graphe de spécification de DefinitionGraph. . . . .	78
3.34	Graphe de spécification de RestrictionGraph. . . . .	78
3.35	Graphe de spécification de lf. . . . .	79
3.36	Graphe de spécification de Then. . . . .	79
3.37	Graphe de spécification de Context. . . . .	80
3.38	Graphe de spécification de attch. . . . .	80
3.39	Graphe de spécification de blng. . . . .	81
3.40	Graphe de spécification de conf. . . . .	81
3.41	Graphe de spécification de csubt. . . . .	81
3.42	Graphe de spécification de def. . . . .	82
3.43	Graphe de spécification de extn. . . . .	82
3.44	Graphe de spécification de from. . . . .	82
3.45	Graphe de spécification de intn. . . . .	82
3.46	Graphe de spécification de is-elt. . . . .	83
3.47	Graphe de spécification de proj. . . . .	83
3.48	Graphe de spécification de pnt. . . . .	83
3.49	Graphe de spécification de ref. . . . .	84
3.50	Graphe de spécification de rep. . . . .	84
3.51	Graphe de spécification de rstrct. . . . .	84
3.52	Graphe de spécification de rsubt. . . . .	85
3.53	Graphe de spécification de sntx. . . . .	85
3.54	Graphe de spécification de sub. . . . .	85
3.55	Graphe de spécification de type. . . . .	85
3.56	Règle de définition d'une relation transitive. . . . .	88
3.57	Règle de définition d'une relation symétrique. . . . .	88
3.58	Règle de définition d'une relation anti-symétrique. . . . .	89
3.59	Règle de définition d'une relation réflexive. . . . .	89
3.60	La proposition "Marie pense que Pierre l'aime". . . . .	90

3.61	L'intension du contexte des pensées de Marie. . . . .	91
3.62	L'extension du contexte des pensées de Marie. . . . .	91
3.63	La proposition "Marie pense que Pierre est beau". . . . .	91
3.64	La proposition "Marie pense que Pierre pense qu'il est beau". . . . .	91
3.65	Le contexte des pensées de Marie. . . . .	92
3.66	Le contexte des pensées de Marie au sujet des pensées de Pierre. . . . .	92
3.67	Le contexte des pensées de Pierre. . . . .	92
3.68	Le treillis de contextes formels. . . . .	95
3.69	La partie langage de la pyramide des modèles. . . . .	95
4.1	Spécialisation et instanciation des types. . . . .	98
4.2	Le langage est utilisé à tous les niveaux. . . . .	98
4.3	Hierarchie des types du méta-métamodèle. . . . .	99
4.4	Les liens entre langage et méta-métamodèle. . . . .	100
4.5	Le méta-métamodèle (formalisme UML). . . . .	100
4.6	Hierarchie des types du méta-métamodèle. . . . .	102
4.7	Le métamodèle (formalisme UML). . . . .	103
4.8	Exemple de document XML et graphes conceptuels. . . . .	110
4.9	'Quels sont les livres écrit par Georges Gardarin?'. . . . .	110
4.10	'Quels sont les livres où apparaît Georges Gardarin?'. . . . .	111
4.11	Un ensemble de graphes à interroger. . . . .	112
4.12	Les graphe résultats $G' = \sigma(G, q_1)$ . . . . .	112
4.13	Les chats assis sur un tapis. . . . .	112
4.14	La structure des contextes formels. . . . .	114
4.15	Graphe de spécification de la fonction $\sigma$ . . . . .	116
4.16	Une demande d'adhésion. . . . .	118
4.17	Modèle de l'activité Traiter une demande. . . . .	119
4.18	La mission de la coopérative. . . . .	120
4.19	"Qu'est-ce qu'une activité de support?". . . . .	120
4.20	"Qu'est ce qu'une mission?". . . . .	121
4.21	Trois niveaux de la pyramide. . . . .	121
4.22	La partie méta-métamodèle et métamodèle de la pyramide des modèles. . . . .	122
5.1	Multi-classification. . . . .	125

5.2	Multi-classification, migration et connaissance partielle. . . . .	126
5.3	Relations définies au niveau instance. . . . .	127
5.4	Mission vue comme un type. . . . .	127
5.5	Mission vue comme une instance. . . . .	127
5.6	Cardinalités : minimum 1, maximum 1. . . . .	128
5.7	Cardinalités : minimum 1, pas de maximum. . . . .	129
5.8	Cardinalités : minimum i, maximum j. . . . .	129
5.9	Sous type avec cardinalité 1..n. . . . .	130
5.10	Partage d'activités. . . . .	131
5.11	Partage d'activités. . . . .	132
5.12	Contrainte de clé unique. . . . .	133
5.13	Contrainte de non nullité. . . . .	133
5.14	Contrainte d'appartenance à un domaine. . . . .	134
5.15	Contrainte de dépendance fonctionnelle. . . . .	134
5.16	Contrainte de dépendance fonctionnelle inter relation. . . . .	135
5.17	Contrainte de dépendance d'inclusion. . . . .	136
5.18	Contrainte d'implication. . . . .	136
5.19	Contrainte de migration. . . . .	136
5.20	Contrainte de migration. . . . .	137
5.21	Contrainte de migration. . . . .	137
5.22	Contrainte de couverture. . . . .	138
5.23	Contrainte de disjonction. . . . .	138
5.24	Contrainte de contexte. . . . .	139
5.25	Contrainte d'implication d'informations négatives. . . . .	139
5.26	Contrainte de restriction de domaine. . . . .	140
5.27	Contrainte d'exclusivité. . . . .	140
5.28	Contrainte d'exclusivité. . . . .	141
5.29	Les modèles de CommonKads. . . . .	143
5.30	Relations entre les composantes de "domain knowledge". . . . .	145
5.31	Hiérarchie des types de CommonKADS. . . . .	148
5.32	Graphe de spécification de Ontology. . . . .	149
5.33	Graphe de spécification de Concept. . . . .	149
5.34	Graphe de spécification de Attribute. . . . .	150

5.35	Graphe de spécification de Expression. . . . .	150
5.36	Graphe de spécification de Structure. . . . .	151
5.37	Graphe de spécification de Relation. . . . .	151
5.38	Graphe de spécification de Inference. . . . .	152
5.39	Graphe de spécification de Task. . . . .	152
5.40	Graphe de spécification de TaskDefinition. . . . .	153
5.41	Graphe de spécification de TaskBody. . . . .	153
5.42	Graphe de spécification de PSM. . . . .	154
5.43	Premier niveau de la hiérarchie de type du noyau. . . . .	156
5.44	Hiérarchie de types de l'ontologie Noyau. . . . .	158
5.45	Hiérarchie de types de l'ontologie Système. . . . .	161
5.46	Hiérarchie de types de relation de l'ontologie ST (System Thinking). . . . .	170
5.47	Le cycle de production. . . . .	175
A.1	Exemple de graphes conceptuels. . . . .	190
A.2	Notation des concepts individuels. . . . .	192
A.3	Notation des concepts génériques. . . . .	192
A.4	Lien de coréférence. . . . .	196
A.5	Proposition et contexte emboîtés. . . . .	197

# Liste des Tables

- 1.1 Exploitation des informations en fonction de leur forme. . . . . 11
- 2.1 Synthèse des formalismes de structure. . . . . 34
- 2.2 Synthèse des formalismes de dynamique. . . . . 49

*à Adrien et Arthur*  
*à Brigitte*

# Remerciements

Je tiens à exprimer mes plus vifs remerciements à Rudolf Keller, professeur à l'université de Montréal et Guy Mineau, professeur à l'Université Laval pour avoir accepté de diriger mes travaux de recherche. Tout au long de ce travail ils ont su me guider et m'ont fait bénéficier de leurs pertinents conseils.

Je remercie Esma Aïmeur pour avoir accepté de présider ce jury de thèse et Houari Sahraoui pour avoir accepté d'y participer.

Je voudrais remercier vivement John Sowa pour le grand honneur qu'il me fait d'être l'examineur externe. Ses travaux de recherche sur les graphes conceptuels sont à l'origine de cette thèse. Sa créativité et sa rigueur scientifique ont été pour moi une grande source d'inspiration.

Les travaux présentés dans cette thèse ont été initiés au Centre de Recherche et de Développement du Groupe Conseil DMR Inc. à Montréal. Je tiens à remercier particulièrement Jean-Marc Proulx et Benoît Guay pour m'avoir intégré au projet Macroscope et m'avoir supporté dans ce projet de doctorat. Ils m'ont fourni un sujet d'étude des plus intéressants dans un environnement stimulant.

Je voudrais également remercier tous les membres de l'équipe du Référentiel Méthodologique avec lesquels j'ai travaillé chez DMR pendant 7 ans. Un merci particulier à Pierre Jarest, Normand Leduc et Jean-François Lamy pour nos nombreuses discussions, sans oublier Mario Perron pour nos longs échanges sur les graphes conceptuels.

Cette thèse se termine alors que je suis professeur au Service de l'Enseignement des Technologies de l'Information à l'École des HEC. Je voudrais remercier ici mes collègues pour la confiance qu'ils m'ont accordée lors de mon recrutement en juin dernier. Ils ont su également me motiver et m'encourager dans cette dernière période de rédaction. Un merci particulier à Jean Talbot, directeur du service, pour son soutien constant.



Je remercie profondément mes parents pour m'avoir inculqué le goût de l'effort et la passion des études. Ce sont des valeurs que j'espère pouvoir transmettre à mes fils.

Enfin, je remercie mes amis et ma famille qui m'ont beaucoup aidé tout au long de ce doctorat. Merci à Adrien et Arthur pour leurs sourires et leur bonne humeur et pour les doux moments qui m'ont permis de m'échapper un peu. Merci à Brigitte pour sa compréhension, son support et tant d'autres choses.

# Introduction

Charnel Havens, Chief Knowledge Officer de la compagnie *EDS* (Electronic Data Systems), présente, dans un article [41] paru dans *CIO Canada*, les enjeux de la gestion de la connaissance d'une entreprise.

*With a huge portion of a company's worth residing in the knowledge of its employees, the time has come to get the most out of that valuable corporate resource - by applying management techniques.*

Dans ce même article, Charnel Havens rapporte qu'une étude menée en 1995 a montré que les professionnels passaient environ 60% de leur temps à recueillir et à valider de l'information, 18% de leur temps en travail effectif, et enfin 22% de leur temps à assister à des réunions, aller chercher un café, etc. Dans le cas d'*EDS*, Charnel Havens calcule que diminuer d'un tiers le temps passé à collecter et valider l'information représenterait un gain de temps d'une valeur de plus d'un milliard de dollars pour la compagnie. Il est donc nécessaire de travailler aux moyens qui pourraient être utilisés pour réduire le temps passé à chercher et valider l'information.

Une autre problématique rencontrée par les entreprises est la perte de compétences liée aux départs d'employés expérimentés, départs dus à la mise à la retraite, à diverses incitations à quitter l'entreprise ou tout simplement dus au roulement de personnel. Il est donc nécessaire de travailler aux moyens qui pourraient être utilisés pour capitaliser le savoir de ces employés expérimentés.

Souvent lié à la problématique précédente, le transfert de connaissances ou formation des nouveaux employés est un défi pour beaucoup d'entreprises. Les nouveaux employés doivent être opérationnels le plus rapidement possible. Les temps de formation doivent donc être les plus courts possibles, et ceci exige un apprentissage "juste à temps" et "juste suffisant". Il est donc nécessaire de travailler aux moyens qui pourraient être utilisés pour raccourcir le temps de formation et fournir l'aide nécessaire

à la réalisation au moment de l'exécution de la tâche.

Charnel Havens dans la conclusion de son article [41] résume bien le consensus actuel sur la valeur et les enjeux liés à la connaissance corporative, et il donne la clé pouvant assurer la mise en valeur des connaissances corporatives : un système de gestion de la connaissance.

*In conclusion, think about whether your company is experience-rich and leverage-poor. If you are wasting knowledge energy, throwing away your intellectual capital, then you are ignoring one of your company's most valuable assets. A knowledge management system can be the key to putting those assets to work.*

Les connaissances et la capacité d'une organisation à apprendre sont vues comme la principale source d'avantage compétitif [63], et le défi à relever est la gestion de cette connaissance corporative. La gestion de la connaissance nécessite l'acquisition, l'entreposage et la dissémination de la connaissance acquise par l'organisation [71]; et les systèmes informatiques sont probablement le moyen de mettre en oeuvre les mémoires d'entreprise [73] qui atteindront ces objectifs.

Nous donnerons pour exemple le *Groupe Conseil DMR Inc.*, pour lequel nous avons travaillé, et au sein duquel nous avons débuté ce travail de recherche. Le *Groupe Conseil DMR Inc.* a pris conscience de l'importance stratégique que représentaient l'expérience et les connaissances acquises par ses consultants dans le domaine des technologies de l'information. Afin de tirer tous les bénéfices de cette expertise, le *Groupe Conseil DMR Inc.* a mis sur pied le projet de recherche *Le Macroscopie Informatique* [25] dont l'objectif était la gestion de cette expertise et des produits en découlant. Le projet *Macroscopie* est le contexte dans lequel notre recherche a été initiée. Ce projet nous a permis de mesurer l'importance d'une approche gestion de connaissances dans le domaine des technologies de l'information, d'identifier les problèmes liés au développement d'un système de gestion de la connaissance et de proposer et valider une approche générique pour le développement de tels systèmes.

Avant de présenter plus précisément le problème auquel s'attaque ce travail de recherche, nous allons d'abord préciser la notion de connaissance corporative et les limitations des approches actuelles, puis nous présenterons les différents modèles nécessaires à la représentation de cette connaissance ainsi que leurs différentes utilisations possibles.

## La connaissance corporative

Qu'il s'agisse de stocker de l'information, de capturer la connaissance des seniors, ou de transférer cette connaissance aux juniors, il est nécessaire d'avoir un système de gestion de la connaissance corporative. La connaissance corporative, c'est l'ensemble des savoir-faire de l'entreprise, c'est-à-dire ses processus d'affaires, ses procédures, ses politiques (mission, règlements, normes) et ses données (ventes, achats, salaires, etc.).

Pour ces dernières, les données, les entreprises disposent aujourd'hui de systèmes de gestion de données. Ce sont les systèmes de gestion de fichiers, les systèmes de gestion de bases de données hiérarchiques, les systèmes de gestion de bases de données relationnelles (SGBDR), les systèmes de gestion de bases de données orientées objet (SGBDO) ou encore des systèmes plus sophistiqués pour colliger ou analyser les données, tels que les "data warehouse" et les systèmes de "data mining". Pour la partie savoir-faire, les entreprises ne disposent pas de systèmes spécialisés. Elles utilisent, pour les plus avancées, des systèmes de gestion documentaire, pour gérer les documents décrivant leurs processus d'affaires, leurs procédures ou encore leurs politiques corporatives.

Mais les documents par eux-mêmes ne sont pas suffisants pour assurer une gestion de la connaissance. Si la version électronique d'un document est mieux que la version papier dans le sens où la version électronique permet la recherche de chaînes de caractères, elle ne supporte aucun modèle et aucune sémantique. Seul un être humain peut interpréter, par exemple dans la description d'un processus, la numérotation des sections et sous-sections comme la décomposition du processus en activités et sous-activités.

La définition d'un modèle de données et l'utilisation d'un système de gestion de bases de données, comme les systèmes relationnels ou orientés objet, permettrait déjà une meilleure gestion de la connaissance. Les modèles de données obligent à une structuration des connaissances et les langages d'interrogation permettent d'exploiter efficacement les données. Cependant l'ensemble des savoir-faire de l'entreprise, objets d'intérêt, processus, procédures ou encore politiques sont de nature trop différente pour être supportés directement par les systèmes de gestion de bases de données. Trop de connaissances restent implicites, les textes ne sont pas analysés et les contraintes ou règles ne sont pas représentées.

## Les modèles

Le développement d'un système de gestion de connaissances doit être basé sur un ensemble de modèles pour représenter toutes les formes de connaissance. Nous distinguons le modèle de structure, le modèle de dynamique et le modèle de contrainte. Pour la partie statique ou structurelle de la connaissance corporative, le modèle de structure permet de représenter les objets d'intérêt de l'entreprise ainsi que les relations que ces objets entretiennent entre eux. Pour la partie dynamique, le modèle de dynamique permet la représentation des processus ou procédures de l'entreprise. Que ce soit au niveau des structures ou de la dynamique, il faut pouvoir représenter les contraintes ou règles qui les régissent. Le modèle de contraintes permet de les représenter. Mais outre ces raisons d'être des différents types de modèles, d'autres utilisations peuvent en être faites.

Les modèles de structure, qui servent à stocker la connaissance, peuvent être utilisés dans un but didactique pour expliquer les modèles ; ou encore la connaissance qu'ils représentent peut être exploitée pour convertir un modèle d'un formalisme à un autre (par exemple passer d'un modèle entité-relation à un modèle objet).

Les modèles de dynamique qui permettent la représentation et le stockage des processus peuvent, dans un contexte d'apprentissage, être utilisés pour enseigner et expliquer les processus. Ils peuvent également fournir les informations nécessaires à la simulation ou à l'exécution d'un processus (workflow manager). Dans le cadre d'un environnement électronique d'aide à la tâche (Electronic Performance Support System), les modèles de dynamique peuvent fournir les connaissances contextuelles d'aide nécessaires à l'exécution d'une tâche.

Les modèles de contraintes qui représentent les contraintes ou règles qui régissent les connaissances, peuvent servir à la validation des connaissances. Ces mêmes modèles dans un contexte d'acquisition de connaissances fournissent les informations pour expliquer les contraintes, c'est-à-dire, donner les raisons pour lesquelles des connaissances ne sont pas acceptables, i.e., violent la sémantique du domaine modélisé.

## Les niveaux d'utilisation des modèles

Nous venons de voir que les différents types de modèles nécessaires à la représentation des connaissances peuvent être utilisés de différentes manières. Selon ces types d'utilisation, nous pouvons distinguer trois niveaux d'utilisation de représentation des mêmes modèles : le niveau données, le niveau méta et le niveau métaméta.

Le niveau données correspond à l'utilisation des modèles conformément à leur raison d'être, c'est-à-dire, pour stocker et valider les connaissances, pour stocker et valider le comportement, pour simuler ou exécuter la dynamique et pour stocker les contraintes. Dans un contexte de mémoire d'entreprise, le niveau données supporte la tâche. Le niveau données fournit les informations nécessaires à un ou une employée pour faire son travail.

Le niveau métamodèle correspond à l'utilisation des modèles, non comme modèles pour stocker des connaissances mais pour fournir des informations. Ce niveau permet d'expliquer les structures, la dynamique et les contraintes. Le niveau méta supporte l'apprentissage des tâches par un ou une employée.

Le niveau méta-métamodèle correspond à l'utilisation des modèles comme informations sur les modèles. Ces informations permettent de convertir des modèles, de comparer et d'intégrer des modèles, des dynamiques et des contraintes ou encore de valider la cohérence d'un ensemble de contraintes.

## **Le problème et notre approche**

Comme nous venons de le mentionner, la gestion des connaissances corporatives nécessite (i) la définition de plusieurs modèles : modèles de structure, de dynamique et de contrainte, et (ii) la manipulation de ces modèles selon trois niveaux d'utilisation : données, méta et métaméta. Ces exigences aboutissent à la nécessité d'utiliser des formalismes de modélisation puissants et complets. Malgré l'existence de nombreux formalismes de modélisation, l'étude préliminaire que nous avons menée [35] nous amène à constater qu'aucun de ces formalismes ne permet de représenter tous les modèles selon ces trois niveaux de représentation, de façon totalement intégrée.

Or, le problème le plus important est celui de l'uniformité. En effet, la construction d'un système de gestion de mémoire d'entreprise ne peut se faire que sur un modèle uniforme intégré et non sur un ensemble de modèles décrits dans des formalismes différents avec tous les problèmes de traduction et de pertes d'information que cela implique dans les échanges entre modèles, supposant que tous ces modèles soient compatibles.

Nous nous sommes donc fixés comme objectif pour notre recherche de proposer un formalisme uniforme répondant à l'ensemble des besoins de représentation d'une mémoire d'entreprise. Plus précisément, nous avons deux objectifs : (i) montrer que les graphes conceptuels permettent de définir un modèle uniforme de représentation

de la connaissance et (ii) montrer que ce modèle uniforme permet de modéliser tous les types de connaissances nécessaires à la constitution d'une mémoire d'entreprise. Ce formalisme doit permettre de représenter l'ensemble des modèles et des niveaux de modélisation présentés précédemment. Ce travail de recherche a été initié dans le cadre du projet *Macroscope* du *Groupe Conseil DMR Inc.*, qui nous a fourni un environnement pertinent pour la validation de notre formalisme.

L'examen détaillé des différents formalismes de modélisation nous a conduits à retenir le formalisme des graphes conceptuels [70] comme formalisme de notre travail. Ce formalisme, basé sur les graphes existentiels de C.S Peirce et sur les réseaux sémantiques, offre la puissance d'expression et la flexibilité requises. Cependant ce formalisme n'intègre dans le standard que la représentation du modèle de structure.

Notre travail de recherche a consisté alors à étendre et à intégrer à ce formalisme de base les ontologies pour les différents types de modèle, afin de constituer un modèle uniforme pour la modélisation et la évangélisation d'une mémoire d'entreprise.

## **Contributions majeures de la thèse**

Les contributions majeures de cette thèse sont :

1. Nous avons effectué un travail exhaustif de modélisation du formalisme des graphes conceptuels avec les graphes conceptuels eux-mêmes. Ceci a montré l'adéquation des graphes conceptuels pour la métamodélisation.
2. Nous avons proposé une formalisation de la notation et de la manipulation des concepts par une approche de métamodélisation, levant ainsi toute ambiguïté concernant l'utilisation des concepts.
3. Nous avons montré que le travail théorique de modélisation du formalisme des graphes conceptuels et les ontologies émergeant de notre recherche constituent une base solide sur laquelle il est possible de développer un système de gestion de mémoire d'entreprise.
4. Nous avons également montré que l'utilisation du langage va au-delà de son utilisation dans la modélisation des mémoires d'entreprise et qu'il pourrait être utilisé dans d'autres domaines de modélisation d'expertise, notamment en modélisant à l'aide du langage développé dans cette thèse, les primitives de modélisation de CommonKads.

5. Nous avons validé le modèle uniforme proposé dans cette thèse dans plusieurs cas concrets de représentation de connaissances corporatives.

L'originalité de ces travaux tient en grande partie à l'utilisation d'un même formalisme, les graphes conceptuels, pour la modélisation des différents niveaux de connaissances. En effet, dans la communauté des graphes conceptuels, aucun travail de cette nature n'a été réalisé à notre connaissance. Enfin, l'originalité et la contribution des travaux de recherche présentés dans cette thèse ont été reconnues par la communauté à travers les publications dont ils ont fait l'objet [38, 36, 35, 52, 37, 34].

### **Organisation de la thèse**

Le présent document est organisé comme suit. Le premier chapitre introduit le contexte dans lequel s'inscrit ce travail de recherche. Le deuxième chapitre présente l'état de l'art concernant les différents formalismes pour la représentation des différents types de connaissances. Le troisième chapitre est le cœur de notre thèse ; il définit le langage du modèle uniforme. Dans le quatrième chapitre sont définis le méta-métamodèle et les métamodèles du modèle uniforme. Le cinquième chapitre détaille les évaluations théoriques et pratiques que nous avons effectuées, et enfin, nous concluons sur les perspectives qu'ouvre le travail présenté dans cette thèse, tant au niveau des applications que des avenues de recherche.



# Chapitre 1

## Contexte

Notre travail de recherche s'inscrit dans le cadre de la représentation des connaissances appliquée à la gestion des connaissances dans les entreprises. La gestion des connaissances dans les entreprises couvre principalement les connaissances liées aux objets et processus d'affaires de l'entreprise.

Il y a actuellement un consensus sur la valeur de la connaissance corporative. Cette connaissance est fondamentale; un employé doit connaître le processus de travail de l'entreprise, un dirigeant doit anticiper les tendances du marché, un chercheur doit être au fait de l'état de l'art et faire de la veille technologique. La connaissance corporative est un tout constitué de stratégies, visions, règles, procédures, directives, traditions et ressources humaines. Les connaissances et la capacité d'une organisation à apprendre sont vues comme la principale source d'avantage compétitif [63], et le défi à relever est la gestion de cette connaissance corporative [41] qui passe par la mémorisation de la connaissance, son intégration et entreposage et sa diffusion au travers de l'organisation.

La connaissance doit être capitalisée et gérée dans des mémoires d'entreprises, dans le but d'assurer une standardisation ainsi qu'une cohérence. La gestion de la connaissance nécessite l'acquisition, l'entreposage et la dissémination de la connaissance acquise par l'organisation [71], et les systèmes informatiques sont probablement le moyen de mettre en oeuvre les mémoires d'entreprise [73] qui atteindront ces objectifs.

Notre travail de recherche a été initié dans le cadre d'un projet de recherche plus vaste, *Le Macroscopie Informatique* [25], projet de recherche à l'initiative du *Groupe Conseil DMR Inc.*, un grand fournisseur de services liés aux technologies de l'infor-

mation. Le Macroscopie, résultat de ce projet, est un ensemble de méthodes, processus, outils logiciels et outils d'apprentissage qui permet aux entreprises de maîtriser l'évolution et la gestion des technologies de l'information. Durant le projet, de nombreux produits décrivant les méthodes et processus tels que des guides, des matériels d'apprentissage et d'auto-apprentissage, des matériels de références, des gabarits de documents et des vidéos ont été développés. Ces différents produits sont entreposés dans une mémoire d'entreprise dont la structure de données et les fonctionnalités permettent une consultation aisée, l'adaptation à des besoins spécifiques d'une organisation et sa maintenance à des coûts acceptables [24]. Cette mémoire d'entreprise, appelée Référentiel méthodologique, joue un rôle fondamental. Elle capture, entrepose [36], retrouve et diffuse [38] au travers de l'organisation tous les processus d'ingénierie de conseil et de développement de systèmes ainsi que la connaissance liée à ces processus, produite par des experts dans le domaine des technologies de l'information.

Avant de présenter les mémoires d'entreprise, afin de bien comprendre les objets manipulés dans l'entreprise ainsi que leurs différents usages, nous allons préciser la notion d'information. Nous ferons également un bref historique de l'évolution de l'informatisation des entreprises : des premiers logiciels aux serveurs de connaissance.

## 1.1 La notion d'information

Une information peut prendre différentes formes, ces différentes formes ayant des potentiels d'utilisation fort différents. Nous pouvons distinguer au moins cinq formes :

- papier (hardcopy),
- fichiers texte (ASCII),
- documents structurés,
- bases de données,
- bases de connaissances.

Chacune de ces formes a un potentiel d'utilisation, ce potentiel étant moindre avec une version papier et augmentant jusqu'à son apogée avec une représentation symbolique.

**Papier (hardcopy).** Seul un être humain peut lire et comprendre la structure d'un document papier et le sens du texte et des graphiques. Aucun traitement n'est possible par un ordinateur. Les ciseaux et la colle sont les moyens d'édition. Le document papier doit être numérisé pour être transformé en un fichier texte.

**Fichiers texte (ASCII).** Les fichiers texte permettent une édition électronique, une vérification assistée par l'ordinateur de l'orthographe et de la grammaire. Les fonctions de recherche permettent de retrouver des chaînes de caractères. Il n'y a pas de représentation de la syntaxe et encore moins de la sémantique. La structure de document est implicite (1, 1.1, ...) et interprétée par le lecteur.

**Documents structurés.** Les documents structurés permettent, comme les fichiers texte, une édition électronique et une vérification assistée de l'orthographe et de la grammaire. De plus, des normes (SGML [2], LaTeX [39]) permettent la représentation explicite de la structure d'un document. La présentation est dans une certaine mesure indépendante du contenu. Beaucoup d'entreprises aujourd'hui ont leur connaissance corporative dans des documents structurés, et les plus évoluées utilisent des gestionnaires de documents. Cependant ces gestionnaires de documents ne gèrent pas la sémantique, si ce n'est en offrant des recherches par des mots-clés (avec dictionnaires) qui permettent aux utilisateurs de faire une recherche dans la base de documents. D'autres systèmes offrent des recherches dans le texte des documents avec des résultats désastreux (plusieurs dizaines voire centaines de réponses non pertinentes) lorsque l'on ne sait pas précisément ce que l'on cherche.

**Bases de données.** L'étape suivante est l'utilisation de bases de données pour stocker la connaissance corporative. Dans ce cas la connaissance est structurée par le schéma de la base. De plus, l'intégrité référentielle peut être assurée par le système de gestion de bases de données. La présentation des données est totalement indépendante de la structuration. Les mêmes informations peuvent être montrées différemment ; des calculs, des statistiques peuvent être effectués sur ces données. Cependant, il y a des limitations car une partie de la connaissance reste implicite ; les phrases ne sont pas analysées, elles peuvent donc référer à des objets qui n'existent pas ou n'existent plus dans la base de données. Les contraintes ne sont pas facilement accessibles pour être utilisées pour former les personnes chargées de l'acquisition des connaissances, par exemple. La recherche dans une base de données nécessite la connaissance de la structure de la base et d'un langage de requêtes, comme SQL. Les bases de données sont cependant le premier pas vers les bases de connaissances.

**Bases de connaissances.** Les bases de connaissances représentent de manière explicite la connaissance. Sous des représentations formelles, toutes les informations

peuvent être analysées y compris le langage naturel. La connaissance est structurée selon un métamodèle. Les bases de connaissances offrent la possibilité de représenter les contraintes et règles d'inférence. Il est donc possible d'inférer des connaissances nouvelles à partir du corpus existant.

Le tableau 1.1 ci-dessous présente les compétences requises par un être humain et les possibilités offertes par l'ordinateur pour exploiter des informations représentées sous les cinq différentes formes discutées ci-dessus.

	<b>Humain</b>	<b>Ordinateur</b>
<b>Papier</b>	Lecture	-
<b>Fichier texte</b>	Lecture	Recherche de chaînes de caractères
<b>Document structuré</b>	Lecture	Recherche de sections
<b>Bases de données</b>	Connaissance de la structure des tables	Recherche basée sur langage SQL
<b>Connaissance</b>	Connaissance du vocabulaire	Recherche basée sur langage naturel contrôlé

TAB. 1.1: Exploitation des informations en fonction de leur forme.

## 1.2 L'informatisation des entreprises

Depuis le début des années soixante, les entreprises automatisent de plus en plus les tâches administratives. L'informatisation des entreprises a évolué des tâches administratives routinières (gestion de stock, calcul de paie) aux tâches administratives les plus sophistiquées (aide à la décision des dirigeants). Dans cette évolution nous pouvons distinguer les outils informatiques suivants :

- logiciels,
- infocentre,
- entrepôt de données (data warehouse),
- recherche de patrons (data mining),
- gestionnaire de documents,
- registre d'expérience,
- serveur de connaissance.

**Logiciels.** Les logiciels spécialisés ont été introduits dans les entreprises pour automatiser certaines tâches administratives comme la comptabilité et la gestion de stock. Les systèmes de gestion de bases de données ont permis une plus grande souplesse et un développement plus rapide des applications.

**Infocentre.** Le nombre de bases de données augmentant considérablement dans les entreprises, le problème d'accès à l'information trop souvent dispersée s'est posé. La notion d'infocentre permet à l'aide de tableaux de bord d'interroger et de présenter les données ou des statistiques sur les données des différentes bases existantes dans l'entreprise. Cependant chaque base de données a sa propre structure, parfois incohérente ou redondante avec d'autres bases.

**Entrepôt de données (data warehouse)** Un entrepôt de données est le résultat du traitement des données des différentes bases existantes. Les données sont traitées afin de se conformer à un modèle commun de données. Dans un entrepôt de données, il y a consolidation des données sans incohérence ou redondance. De plus toutes les données étant conformes à un modèle commun, il est possible de faire des recoupements d'information - ce qui est impossible dans l'infocentre.

**Recherche de patrons (data mining)** La recherche de patrons dans les données est rendue possible par l'entrepôt de données. Les données de l'entreprise étant structurées et traduites dans un modèle commun, il est non seulement possible de l'interroger, mais également de rechercher des structures ou relations non définies par le modèle et de faire émerger de nouveaux modèles.

**Gestionnaire de documents.** Aujourd'hui, la plupart des informations corporatives sont consignées dans des documents, qui vont de la simple note de service au complexe rapport financier annuel. Les disques durs des ordinateurs sont submergés de documents de toute sorte et les gestionnaires de documents ont été développés pour aider à y mettre de l'ordre. Les documents sont classés et indexés conformément aux mots-clés définis par le rédacteur du document ou par un spécialiste de l'indexation. Ces mots-clés sont utilisés pour retrouver les documents.

**Registre d'expérience.** Un registre d'expérience permet d'emmagasiner les leçons apprises de l'expérience, des meilleures pratiques aux plus mauvaises pratiques à ne

pas reproduire. Rares sont les entreprises qui ont implanté un registre d'expérience. En effet, la mise en place d'un registre d'expérience nécessite la définition d'un processus complexe pour la collecte et le traitement des leçons apprises.

**Serveur de connaissances.** Nous avons vu que la plus haute utilité qu'une information peut avoir est lorsqu'elle est sous forme symbolique. Le support pour la diffusion de cette connaissance est le serveur de connaissances. Il est basé sur un modèle sémantique de l'organisation et de son savoir-faire. Le serveur de connaissances grâce à cette intégration sémantique et à sa base de connaissances donne accès aux données brutes, aux données traitées, aux documents, au registre d'expérience et donc à toute la connaissance corporative.

### 1.3 La mémoire d'entreprise

La notion de mémoire d'entreprise est étudiée et discutée depuis un quart de siècle. Cependant l'intérêt pour les mémoires d'entreprise s'est accru ces dernières années avec les possibilités de développer des systèmes informatiques les supportant.

Stein fait dans [71] une revue des concepts relatifs à la notion de mémoire d'entreprise ainsi qu'un certain nombre de recommandations pour les gestionnaires. La mémoire d'entreprise est un cas particulier de la mémoire collective. La mémoire d'entreprise est la mémoire collective des individus de l'entreprise. Elle est composée des mémoires des individus qui partagent des informations au sein de l'entreprise. Mais elle est composée également des processus qui mettent en œuvre cette mémoire. Stein définit le contenu d'une mémoire d'entreprise comme la persistance des messages échangés au sein d'une entreprise. En se basant sur la théorie de l'information, qui définit un échange comme un message qui est encodé par l'expéditeur, transmis via des canaux de distribution et décodé par le receveur, Stein distingue trois types de mémoires collectives : une pour les messages non éphémères, une pour les messages voyageurs et une dernière pour les messages durables. Quand le message est encodé mais pas envoyé immédiatement, cela signifie que l'information contenue dans le message n'est pas éphémère et que par conséquent elle est candidate pour devenir une partie de la mémoire d'entreprise. Le deuxième type de mémoire correspond à un message dont la durée de transmission ou d'échange est non négligeable. Cela démontre que l'information contenue dans le message est d'importance et qu'elle est transmise à de nombreux individus ; cette information est elle aussi candidate pour

devenir une partie de la mémoire d'entreprise. Enfin le dernier type correspond à un message dont la durée d'existence est importante - le message n'est pas détruit après lecture. Ceci signifie que l'information contenue dans le message est susceptible d'être consultée à nouveau ; cette information doit être conservée, elle est aussi candidate pour devenir une partie de la mémoire d'entreprise. Stein propose également un cadre de référence pour classier les différents types d'information d'une mémoire d'entreprise. L'information est classifiée suivant deux dimensions, le niveau d'abstraction (concret versus abstrait) et le niveau normatif (préconisé versus descriptif).

Une autre classification des types de mémoires collectives est proposée par van Heijst, van der Spek et Kruizinga dans [73]. L'existence d'une mémoire d'entreprise repose sur l'acquisition de la connaissance et sur la diffusion de cette connaissance. Les auteurs classifient les types de mémoire suivant le type d'acquisition (active ou passive) et le type de diffusion (active ou passive). La combinaison de ces critères définit quatre types de mémoires d'entreprise : le grenier à connaissance (acquisition et diffusion passive), l'éditeur de connaissance (acquisition passive et diffusion active), l'éponge à connaissance (acquisition active et diffusion passive) et la pompe à connaissance (acquisition et diffusion active).

Aujourd'hui il y a de nombreux travaux autour de la gestion des connaissances (Knowledge Management). Jay Liebovitz a édité un ensemble d'articles [43] présentant l'état de l'art dans le domaine de la gestion des connaissances. On y découvre que la notion de gestion de la connaissance n'est pas un concept nouveau mais plutôt l'intégration de concepts et techniques issus des domaines de l'intelligence artificielle, de l'ingénierie de systèmes d'information, de la réingénierie de processus, de la gestion des ressources humaines et du comportement des organisations. Du point de vue technologie qui nous intéresse dans le cadre de cette thèse, on y découvre qu'il existe de nombreux outils pour supporter le processus de gestion mais qu'il n'existe pas de référentiels pour stocker la connaissance d'une manière conceptuelle et structurée afin de construire une mémoire d'entreprise au sens d'entrepôt de connaissances.

C'est pourquoi aujourd'hui il existe quelques travaux sur les mémoires d'entreprise [23] mais très peu d'exemples de mémoires d'entreprises. Quelques grandes organisations américaines (NASA [5], US Army, US Department of Energy) ont mis en place des mémoires d'entreprise ; ces mémoires concernent principalement les leçons apprises de l'expérience. Cependant les prochaines années verront les besoins augmenter. Le concept de gestion de la connaissance est de plus en plus prisé dans les

entreprises car il apporte des solutions aux problèmes cruciaux de la conservation des connaissances acquises (savoir-faire) par l'entreprise et à la diffusion de la culture d'entreprise dans les entreprises multinationales.

Certains restent critiques, comme Humbert Lesca [47].

*Plutôt que d'une "mémoire d'entreprise" il vaut mieux parler d'un ensemble de mémoires, et cet ensemble a les caractéristiques suivantes :*

- 1. mémoires nombreuses, en nombre pas toujours connu*
- 2. de natures très diverses :*
  - les unes très formalisées*
  - les autres totalement informelles*
- 3. pas toujours identifiées ni localisées*
- 4. non communicantes, ou rarement ou faiblement*
- 5. dont le contenu n'est pas toujours connu*
- 6. dont le contenu est plus ou moins accessible*
- 7. non coordonnées*
- 8. gérées de façon individuelle ou quasi individuelle*

*En d'autres termes, l'expression "mémoire d'entreprise" est actuellement un concept sans existence dans la réalité.*

Cependant nous pensons que les avancées scientifiques et technologiques vont rapidement permettre de capturer un grand nombre de cet "ensemble de mémoires" et de concrétiser l'expression "mémoire d'entreprise". Dans ce but nous proposons dans cette thèse un modèle uniforme pour la création de mémoires d'entreprises. Ce modèle uniforme sera la base permettant de développer des outils de modélisation et méta-modélisation intégrés pour formaliser et capturer facilement ce qui peut se formaliser dans une mémoire d'entreprise.



# Chapitre 2

## État de l'art

Ce chapitre présente l'état de l'art pour la représentation des connaissances nécessaires au développement d'une mémoire d'entreprise. Les connaissances, qui sont liées au savoir-faire d'une entreprise, posent un problème de volume, de complexité et de diversité. Une solution à ce problème est la structuration des connaissances. Cette structuration peut se faire suivant deux axes : un axe horizontal qui définit les types de connaissances : structure et dynamique (2.1) et un axe vertical qui définit les niveaux de modélisation : données, modèle, métamodèle et méta-métamodèle (2.2). Pour chacun des types de connaissances structure (2.3) et dynamique (2.4) nous avons étudié les principaux formalismes de représentation vis-à-vis des niveaux de modélisation. La représentation des contraintes, que nous avons évoquée dans l'introduction, est traitée d'une manière subordonnée aux formalismes de structure et de dynamique et n'a donc pas été traitée d'une manière indépendante.

### 2.1 Les types de connaissances

Sur l'axe horizontal nous distinguons deux aspects. L'aspect *structure* représente la partie statique de la connaissance et l'aspect *dynamique* représente la partie comportementale de la connaissance.

#### **Structure**

L'aspect structure définit les concepts de base et les relations qui peuvent les unir. Dans le cas d'une mémoire d'entreprise, l'aspect structure définit et décrit les objets

d'intérêt pour l'entreprise. Il décrit également les relations qui peuvent exister entre ces objets.

## Dynamique

L'aspect dynamique définit les concepts utilisés pour représenter la dynamique des objets de connaissance. Dans le cas d'une mémoire d'entreprise, nous nous intéressons particulièrement à la représentation et la description des processus d'affaires.

## 2.2 Les niveaux de modélisation

Les principaux travaux de modélisation et métamodélisation ont été effectués par les groupes de travail des différents organismes de normalisation (OMG [40], UML [7], graphes conceptuels [19]) et par les groupes intéressés par l'échange de modèles [18], ou la transformation de modèles [12]. Les besoins de langage commun et les besoins d'échange d'information entre les outils de modélisation ont conduit à une réflexion sur les objets de la modélisation, c'est-à-dire les métamodèles [11]. Aujourd'hui, il y a consensus [40, 7, 18] sur une architecture à quatre niveaux (figure 2.1) :

- méta-métamodèle,
- métamodèle,
- modèle,
- données.

Le *méta-métamodèle* représente le niveau le plus abstrait, il représente le langage de définition des métamodèles. Il définit les notions de base qui vont permettre la représentation de tous les autres niveaux ainsi que lui-même. Des exemples sont MétaClasse, MétaAttribut et MétaOpération dans le cas de UML ; MétaEntité avec (méta)attribut et MétaRelation avec (méta)attribut pour CDIF ; et concept, relation conceptuelle et graphe pour les graphes conceptuels.

Le *métamodèle* définit le langage de représentation ou formalisme des modèles. Des exemples sont Classe, Attribut, Opération pour UML ; Entité, Attribut, Relation pour le formalisme Entité-relation ; et Type de concept, et Type de relation pour les graphes conceptuels.

Le *modèle* définit le langage de représentation du domaine étudié. Des exemples sont Employé, Organisation, Client, Mission.

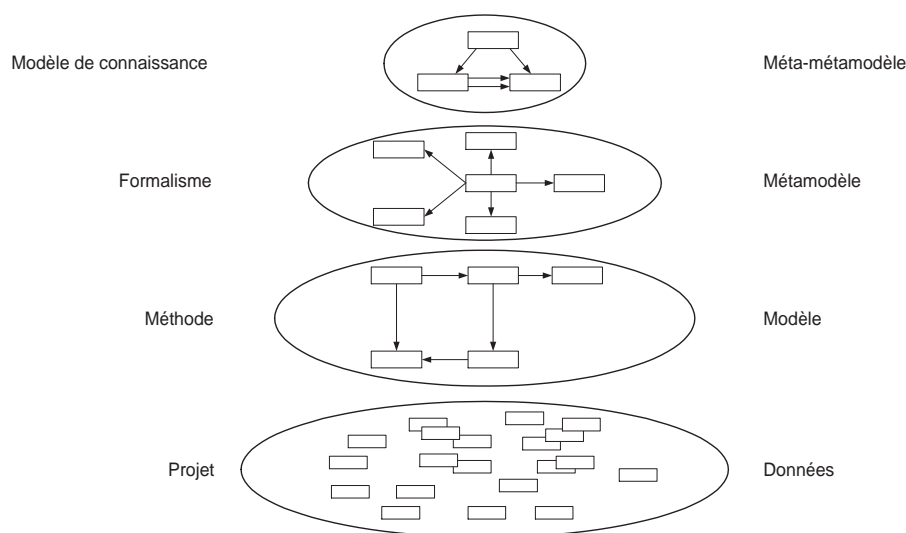


FIG. 2.1: Architecture à quatre niveaux.

Les *données* sont des instances des types du modèle. Elles correspondent aux objets du monde que l'on décrit. Des exemples sont Paul Martin, Hydro-Québec, Fournir de l'électricité.

Pour illustrer la notion de métamodélisation, nous prenons l'exemple d'un projet de développement de système d'information. Sur cet exemple nous allons montrer les différents niveaux de modélisation. Nous les présenterons en commençant par les données et en terminant par le méta-métamodèle ; chaque niveau de modélisation est alors instance du suivant.

### 2.2.1 Les données : le projet

Dans notre exemple, un *projet* est un ensemble d'activités et de biens livrables (spécifications, modèles, code, etc.) produits par ces activités lors du déroulement du projet.

### 2.2.2 Le modèle : la méthode

Le *modèle* du projet ou méthode est la spécification et la description de toutes les composantes d'un projet : activités, ressources, biens livrables, techniques, etc. Les techniques sont des activités nécessitant des habiletés spécifiques (interview, modélisation, programmation, etc.). Les produits du déroulement du projet sont des biens

livrables (documentation, spécification, code, etc.). Chaque bien livrable est constitué d'éléments d'information ; chaque élément d'information documentant un morceau du système. Chaque élément d'information est habituellement produit en utilisant une technique qui définit les procédures et le savoir-faire pour réaliser la tâche nécessaire à sa production. La figure 2.2 présente un exemple de modèle simplifié de projet.

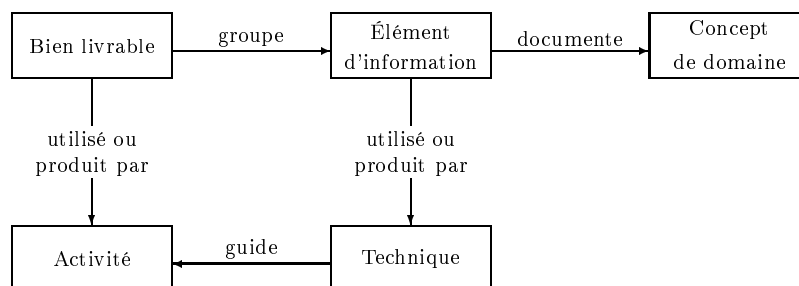


FIG. 2.2: La méthode.

### 2.2.3 Le métamodèle : le formalisme

Le *métamodèle* (figure 2.3) est la description des objets du modèle.

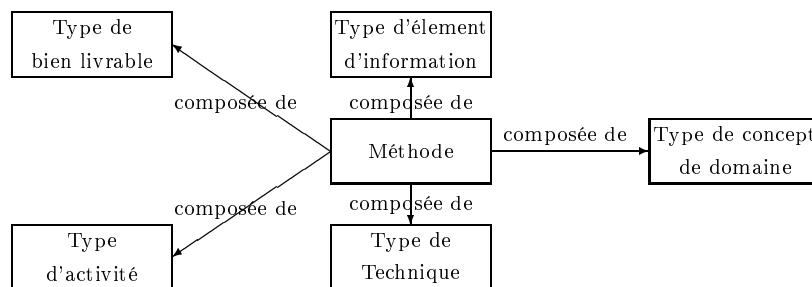


FIG. 2.3: Le formalisme.

Pour les objets de méthode de même nature le modèle définit un type dont les objets de méthode sont des instances. Le métamodèle définit le formalisme utilisé pour décrire les objets de méthode. Dans l'exemple présenté, le métamodèle est celui des graphes conceptuels. Le métamodèle est composé des types de base : **CONCEPT-TYPE** et **RELATION-TYPE** dont sont dérivés les types spécialisés présentés dans la figure 2.3.

## 2.2.4 Le méta-métamodèle : le modèle de connaissance

Le *méta-métamodèle* est la description des objets du métamodèle. Le méta-méta-modèle définit les types dont les objets du métamodèle sont des instances. Dans le cas des graphes conceptuels, les composantes de base sont : **CONCEPT**, **RELATION** et **GRAPHE**. La figure 2.4 présente un méta-métamodèle extrêmement simplifié du formalisme des graphes conceptuels. Nous verrons ce niveau en détail au chapitre 4.

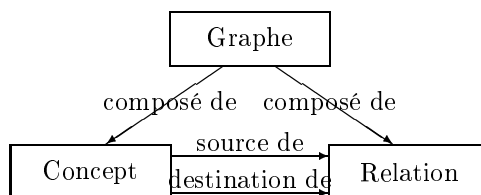


FIG. 2.4: Un méta-métamodèle simplifié.

## 2.3 Formalismes de structure

Cette section présente l'étude menée sur les formalismes de structure et est organisée comme suit. Nous introduisons d'abord les notions de base et définissons les besoins d'une mémoire d'entreprise en terme de représentation de structure. Puis nous détaillons les principaux formalismes de représentation utilisés dans le contexte des entreprises soit : le modèle Entité-Relation Étendu [14], le modèle Orienté Objet [7], le modèle relationnel [16] ainsi que deux formalismes couramment utilisés dans le domaine de la représentation de la connaissance : Classic, un langage de frame [9], et les graphes conceptuels [70]. Pour chacun d'eux nous présentons les notions pertinentes et discutons de leur adéquation à répondre à nos besoins.

### 2.3.1 Notions de base

Les principaux éléments utilisés par les techniques de modélisation sont les catégories ou types, les instances et les relations. Une *catégorie* représente un ensemble de choses qui ont les mêmes propriétés : attributs, relations et comportement. Une *instance* d'une catégorie est une chose qui se conforme à la définition de la catégorie. Une *relation* est une association entre des instances ou des catégories.

Soit EMPLOYÉ la catégorie qui définit ce qu'un employé est. Un employé peut avoir des attributs comme numéro d'employé, nom, etc. Un employé a également une

relation avec la compagnie qui l'emploie. La figure 2.5 illustre cet exemple. EMPLOYÉ et ORGANISATION sont des catégories ; des instances de ces catégories pourraient être Jean, Paul, IEEE, et UNU (Université des Nations Unies) ; 'travaille pour' est une relation entre ces deux catégories.

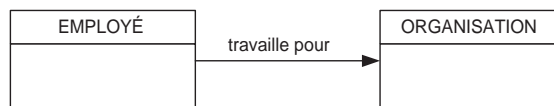


FIG. 2.5: Un exemple de modèle avec deux catégories et une relation.

L'héritage et la classification sont deux autres notions fondamentales utilisées par les techniques de modélisation. L'*héritage* est une relation de type est-une-sortede entre deux catégories. Une catégorie hérite des propriétés de la catégorie parente. La *classification* est la hiérarchie des catégories construites à partir des propriétés héritées. Par exemple, la classification des mammifères est la hiérarchie des espèces du règne animal.

### 2.3.2 Les besoins

Les besoins pour la structuration des données sont certainement ceux qui ont le plus d'importance dans le développement d'une mémoire d'entreprise. Les connaissances corporatives qui sont de nature très variée sont complexes à gérer et à modéliser. Dans cette étude, nous avons retenu trois besoins fondamentaux que le formalisme de structure doit pouvoir satisfaire. Ces trois besoins sont : (i) classification et connaissance partielle, (ii) relations entre catégories et/ou instances et (iii) catégorie ou instance et métamodèle et sont discutés ci-dessous.

#### Classification et connaissance partielle

Les catégories sont définies à partir des propriétés communes de leurs instances. Définir des catégories, c'est définir des critères permettant de différencier les instances les unes des autres. Si le nombre de critères augmente, le nombre de catégories augmente d'une manière exponentielle et ce nombre peut rapidement devenir impossible à gérer, sans oublier que l'adéquation avec la réalité peut être perdue.

Un autre aspect de la classification mais cependant rarement lié à la classification est le problème de la connaissance partielle. Comment classer un objet dont toutes

les propriétés ne sont pas connues ? Par exemple, comment classer la personne Dominique si il y a deux catégories homme et femme et si l'on ne connaît pas le sexe de Dominique ?

Dans la plupart des formalismes, une chose est instance d'une et d'une seule catégorie. Cette limitation impose la définition de toutes les catégories possibles ou concevables où une chose peut être potentiellement classée. Prenons comme exemple un concessionnaire automobile qui désire classer les véhicules à vendre. En se basant sur le nombre de portes, il peut définir les véhicules 2 portes, 4 portes ou les mini-fourgonnettes (Van). En se basant sur le mode de propulsion, il peut définir les véhicules à 2 roues motrices (2RM) ou à 4 roues motrices (4RM). Enfin, en se basant sur l'origine des véhicules, il peut définir trois catégories : Europe, Asie, Amérique. La figure 2.6 montre les hiérarchies basées sur ces trois critères.

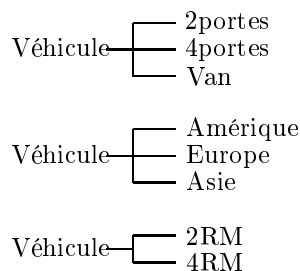


FIG. 2.6: Différentes hiérarchies basées sur trois critères.

Si le vendeur veut classer tous ses véhicules à partir de ces trois critères, toutes les combinaisons sont possibles. Dans cet exemple, cela donne 18 possibilités comme le montre la figure 2.7. Le nombre et le volume de toutes les possibilités peuvent rapidement devenir incontrôlables.

Supposons que le vendeur reçoive deux nouveaux véhicules. La première voiture est une 4 portes avec 2 roues motrices, mais le vendeur ne sait pas où la classer car la voiture a été assemblée en Europe à partir de pièces en provenance d'Asie. Le vendeur pourrait la classer dans la catégorie 2RM 4portes, mais qu'advient-il lorsqu'il connaîtra son origine ? La deuxième voiture est une américaine. Dans quelle catégorie la classer ?

*Pour supporter les besoins de classification dans un contexte de connaissance partielle, le formalisme doit, du point de vue de l'utilisateur, supporter la multi-classification, c'est-à-dire qu'un objet peut être défini comme instance de plus d'une catégorie, et le système doit être capable de faire migrer dynamiquement les instances d'une*

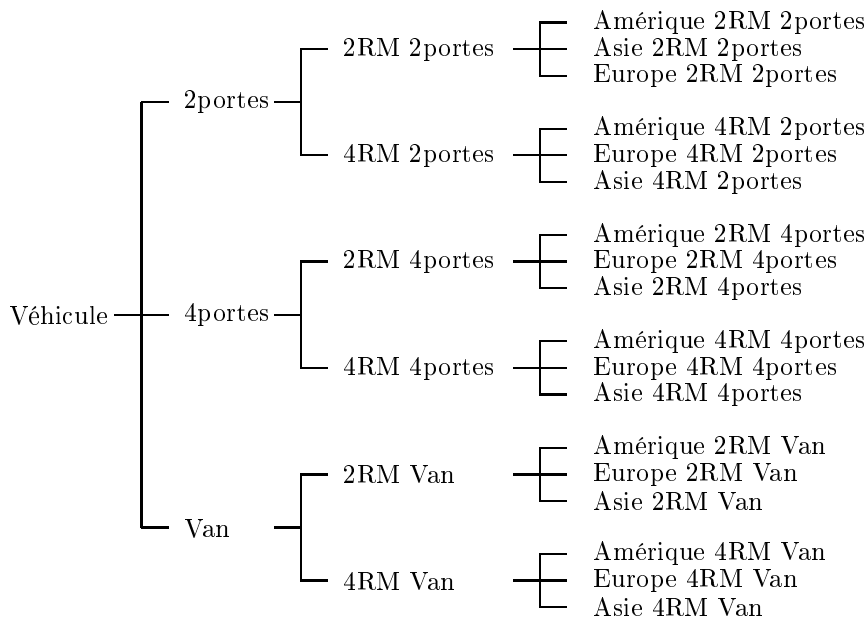


FIG. 2.7: Une hiérarchie basée sur les trois critères.

*catégorie à une autre catégorie.*

### Relations entre catégories et/ou instances

Dans la plupart des techniques de modélisation, les relations sont établies au niveau des catégories et applicables au niveau des instances ; cependant ceci n'est pas toujours suffisant. Dans l'exemple concernant les employés et les organisations, nous voulons distinguer les employés de l'ONU qui travaillent pour l'Université des Nations Unies des autres employés. Soit EMPLOYÉ-ONU la catégorie qui définit les employés de l'ONU. Comme les employés de l'ONU ont les mêmes propriétés, attributs et types de relation que les employés, EMPLOYÉ-ONU est définie comme une sous-catégorie de la catégorie EMPLOYÉ (voir figure 2.8).

Dans la plupart des techniques de modélisation, il est très difficile d'exprimer le fait qu'un employé de l'ONU a une relation avec l'organisation ONU. Les techniques de modélisation expriment la connaissance au niveau catégorie. Les relations sont établies au niveau catégorie ; elles lient des catégories et sont applicables au niveau instance. Dans notre exemple, une catégorie ORGANISATION-ONU, qui n'a qu'une instance (ONU), doit être définie pour pouvoir relier EMPLOYÉ-ONU et ORGANISATION-ONU comme le montre la figure 2.8.



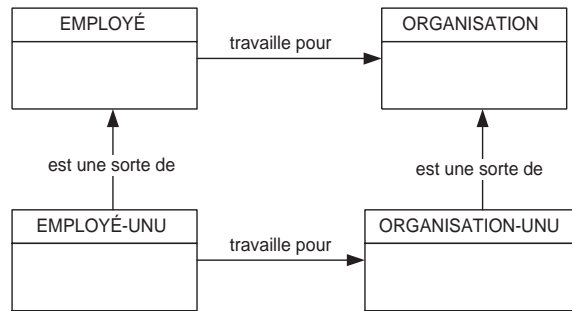


FIG. 2.8: Le modèle complet.

Cependant, ce que nous voulons exprimer est : "Chaque instance de EMPLOYÉ-UNU a une relation 'travaille pour' avec UNU, instance de la catégorie ORGANISATION-UNU.". Ceci correspond à la formule en logique des prédicats du 1er ordre suivante :

$$\forall x, \text{EMPLOYÉ-UNU}(x) \wedge \text{ORGANISATION}(UNU) \Rightarrow \text{travaille-pour}(x, UNU)$$

Mais dans la plupart des techniques de modélisation cela sera traduit par les deux expressions suivantes,

$$\forall x, \exists y, \text{EMPLOYÉ-UNU}(x) \Rightarrow \text{ORGANISATION-UNU}(y) \wedge \text{travaille-pour}(x, y)$$

$$\forall z, \text{ORGANISATION-UNU}(z) \Rightarrow z = UNU$$

où la première expression exprime qu'un employé de l'UNU travaille pour une ORGANISATION-UNU et la deuxième que n'importe quelle ORGANISATION-UNU est l'UNU.

Ces deux formulations sont équivalentes. La première est cependant plus simple et donc préférable car elle évite la création d'une catégorie ORGANISATION-UNU à une seule instance.

*Pour satisfaire ce besoin, le formalisme doit supporter les relations entre catégories et/ou instances, c'est-à-dire qu'une catégorie peut être liée à une autre catégorie ou à une instance, ou que les relations peuvent être directement établies au niveau instance.*

### Catégorie ou instance et métamodèle

Les techniques de modélisation visent à définir des descriptions formelles d'objets ou de notions à partir desquelles on puisse faire des déductions. Cette description

formelle utilise différents types de composantes et différents types de relation. Le métamodèle décrit ces composantes et ces relations. Le métamodèle traite de catégories et de catégories de catégories. Dans la plupart des cas, il est facile de faire la distinction entre catégories et instances. Les catégories fournissent des informations sur les instances, cependant les catégories peuvent être vues comme des instances dans un métamodèle qui donne des informations sur les catégories elle-mêmes.

Dans l'exemple suivant, soit ORGANISATION-FÉDÉRALE une sorte de ORGANISATION. Ceci signifie que ORGANISATION-FÉDÉRALE est une catégorie qui est une sous-catégorie de ORGANISATION. Mais dire que ORGANISATION-FÉDÉRALE est une sous-catégorie de ORGANISATION implique que ORGANISATION-FÉDÉRALE est une instance de la composante de modélisation CATÉGORIE. La figure 2.9 illustre cette situation où ORGANISATION FÉDÉRALE peut être vue, en fonction de la perspective, comme une catégorie ou comme une instance de la catégorie CATÉGORIE.

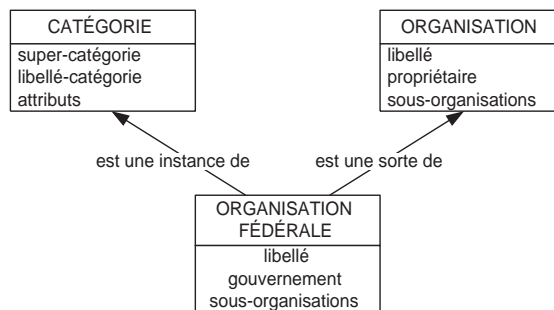


FIG. 2.9: Catégorie ou instance.

De la perspective catégorie, ORGANISATION-FÉDÉRALE est une sous-catégorie de la catégorie ORGANISATION et hérite de ses attributs. Les attributs de ORGANISATION sont : libellé qui nomme l'organisation, propriétaire et sous-organisations. Les attributs hérités sont : libellé, gouvernement qui est une spécialisation de propriétaire et sous-organisations.

De la perspective instance, ORGANISATION-FÉDÉRALE est une instance de CATÉGORIE. Les attributs de catégorie sont : super-catégorie qui définit la position de la catégorie dans la hiérarchie de classification, libellé-catégorie qui nomme la catégorie, et attributs qui donne la liste des attributs de la catégorie. Ces attributs ont les valeurs suivantes pour ORGANISATION-FÉDÉRALE :

- *super-catégorie* : ORGANISATION
- *libellé-catégorie* : ORGANISATION-FÉDÉRALE
- *attributs* : libellé, gouvernement, sous-organisations.

*Pour supporter ce besoin de perspective, le formalisme doit permettre qu'un élément soit vu comme une catégorie ou comme une instance.*

Nous venons de voir des besoins précis auxquels doit répondre le formalisme de représentation de connaissance. Nous analysons ci-dessous chaque formalisme retenu à la lumière de ces besoins spécifiques.

### 2.3.3 Modèle Entité-Relation Étendu

Le modèle Entité-Relation Étendu fut originellement développé par Peter Chen [14] en 1976 et fut étendu par la suite avec les notions d'héritage [26, 27]. Le modèle Entité-Relation Étendu supporte les notions de catégories, appelées entités, et les relations. Il n'y a pas de composante explicite dans le formalisme pour représenter les instances. La représentation de la connaissance se fait au niveau entité. Les entités sont représentées par des rectangles avec le nom de l'entité en haut et les relations par des ovales avec deux arcs qui lient les entités concernées. La figure 2.10 présente le diagramme entité-relation correspondant à "Tout employé travaille pour une organisation".



FIG. 2.10: Entité-Relation Étendu - Tout employé travaille pour une organisation.

**Classification et connaissance partielle.** Il n'y a aucun moyen de représenter les instances dans le formalisme. Une instance est implicitement une instance d'une et d'une seule entité et il n'est prévu aucun mécanisme pour la migration d'instances et pour représenter la connaissance partielle.

**Relation entre catégories et/ou instances.** Comme les instances n'ont pas de représentation dans le formalisme, il est impossible de spécifier des contraintes qui impliquent des instances. La solution communément utilisée dans ce cas est de créer

une entité ayant une et une seule instance, de définir la relation entre les entités et d'ajouter une note explicative comme le montre la figure 2.11.

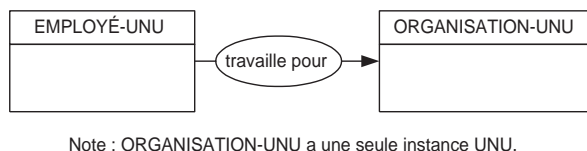


FIG. 2.11: Entité-Relation Étendu - Relation entre catégories et/ou instance.

**Catégorie ou instance.** Le formalisme Entité-Relation Étendu s'applique à un et un seul niveau. Le seul moyen pour exprimer qu'une instance peut être vue comme une catégorie est de faire deux diagrammes, un pour chaque niveau et de les lier par une note explicative (figure 2.12).

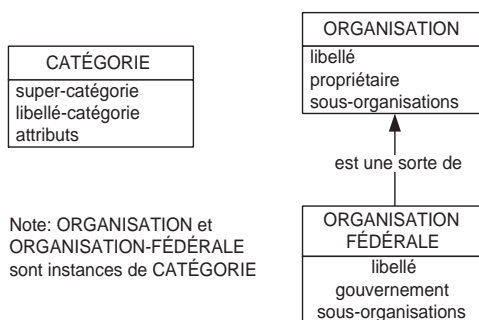


FIG. 2.12: Entité-Relation Étendu - Catégorie ou instance.

### 2.3.4 Formalisme Orienté Objet

Comme formalisme représentatif, nous nous sommes intéressés à UML [7] (Unified Modeling Language) qui est aujourd'hui considéré comme un standard de facto. Ce formalisme a été développé par Grady Booch, Jim Rumbaugh et Ivar Jacobson, et vise l'unification des méthodes Booch [6], OMT [64] et OOSE [44].

Dans le formalisme UML, les instances sont des objets et les catégories des classes, et la classification est supportée. Dans la notation UML, les classes sont représentées par des boîtes rectangulaires, les instances sont représentées comme les classes, mais

avec un libellé souligné. Les relations sont représentées par des arcs joignant les classes. La figure 2.13 présente le diagramme correspondant à "Tout employé travaille pour une organisation."

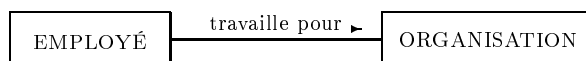


FIG. 2.13: UML - Tout employé travaille pour une organisation.

**Classification et connaissance partielle.** La sémantique d'UML supporte l'héritage multiple et la multiple classification qui pourrait être un support à la connaissance partielle, bien que non évoquée dans UML. L'utilisation de valeurs nulles est également possible pour le support de la connaissance partielle.

**Relation entre catégories et/ou instances.** Les relations sont définies au niveau des classes. Donc comme pour le modèle Entité-Relation Étendu, la solution consiste à définir une classe avec une et une seule instance et d'ajouter une note explicative comme le montre la figure 2.14.

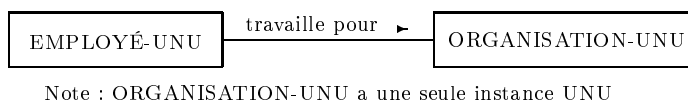


FIG. 2.14: UML - Relation entre catégories et/ou instances.

**Catégorie ou instance.** Les notations (figure 2.15) sont différentes pour représenter les classes et les instances et il n'y a aucun moyen de dire qu'une instance peut être également vue comme une classe. La solution est de définir une classe et un objet avec le même symbole et de laisser l'utilisateur humain l'interpréter.

### 2.3.5 Le modèle relationnel

Le modèle relationnel [16] définit des tuples (instances) et des tables (catégories). La plupart des systèmes de gestion de bases de données relationnelles utilisent le même formalisme pour décrire les tuples et les tables, et les descriptions des tables sont stockées dans un ensemble de tables systèmes appelé métabase. Les relations entre



FIG. 2.15: UML - Catégorie ou instance.

entités sont traduites par des attributs de jointure entre les tables. "Tout employé travaille pour une organisation" est représentée par deux tables (figure 2.16), et dans la table EMPLOYÉ l'attribut OrgId est une clé étrangère qui établit la relation entre un employé et l'organisation pour laquelle il ou elle travaille sachant qu'aucune valeur nulle n'est autorisée afin d'assurer l'intégrité référentielle.

ORGANISATION	OrgId	Nom	...	EMPLOYÉ	EmpId	Nom	OrgId

FIG. 2.16: Modèle relationnel - Tables et attributs de jointure.

**Classification et connaissance partielle.** Un tuple est une ligne d'une et une seule table et il n'y a aucun moyen spécifique pour supporter la connaissance partielle. Cependant le mécanisme de vue relationnelle et l'utilisation de valeurs nulles peuvent être utilisés pour implanter la gestion de la connaissance partielle, mais l'introduction de valeurs nulles apporte souvent des problèmes d'interprétation et peut violer l'intégrité référentielle.

**Relation entre catégories et/ou instances.** Dans le modèle relationnel, les relations sont définies au niveau des instances et implantées en utilisant les attributs de jointure. Pour l'exemple de l'Université des Nations Unies, la solution est de définir une table EMPLOYÉ-UNU avec un attribut de jointure, OrgId, qui a toujours pour valeur l'identifiant de UNU comme l'illustre la figure 2.17.

**Catégorie ou instance.** Les descriptions des tables créées dans un système de gestion de bases de données relationnelles sont normalisées et stockées dans les tables

ORGANISATION	OrgId	Nom	...	EMPLOYÉ-UNU			OrgId
	123	UNU					123
							123

FIG. 2.17: Modèle relationnel - Relations entre catégories et/ou instances.

système (figure 2.18). Au même titre que les autres tables, les tables système peuvent

TABLE	Nom	...	...
	ORGANISATION		
	ORGANISATION FÉDÉRALE		

ATTRIBUT	NomTable	NomAttribut	TypeAttribut
	ORGANISATION FÉDÉRALE	Nom	Texte
	ORGANISATION FÉDÉRALE	Gouvernement	Texte
	...	...	...

ORGANISATION FÉDÉRALE	Nom	Gouvernement	...

FIG. 2.18: Modèle relationnel - Catégorie ou instance.

être manipulées en utilisant le langage de requêtes SQL. Par exemple, utiliser un SELECT dans la clause FROM, comme dans SELECT Gouvernement FROM (SELECT NomTable FROM Attribut WHERE NomAttribut = 'Gouvernement') où le nom de la table de la clause FROM est fourni par une requête, pourrait être une solution. À notre connaissance, aucun système de gestion de bases de données relationnelles n'autorise de telles requêtes.

### 2.3.6 Classic

Classic [9], tels les systèmes de la famille KL-ONE [8], est un système de représentation de la connaissance basé sur les prototypes. Dans Classic les instances sont appelées des *individus* et les catégories sont appelées des *concepts*. Les relations sont

établies entre individus en utilisant les attributs appelés *rôles* dans Classic. La figure 2.19 montre la définition du concept EMPLOYÉ qui représente "Tout employé travaille pour une organisation". EMPLOYÉ est défini comme un sous-type de PERSONNE avec un attribut travaille-pour qui représente la relation avec le concept ORGANISATION.

```
EMPLOYÉ ⇔ (AND PERSONNE(ALL travaille-pour ORGANISATION))
```

FIG. 2.19: Classic - Concepts et relations.

**Classification et connaissance partielle.** Classic supporte la classification multiple. Un individu peut satisfaire la définition de plus d'un concept. Classic supporte également la connaissance partielle grâce à un mécanisme de classification dynamique. Quand un nouvel individu est introduit dans le système, le mécanisme de classification est invoqué pour trouver tous les concepts que le nouvel individu satisfait.

**Relation entre catégories et/ou instances.** Les relations entre individus sont implantées par les rôles. Classic définit des opérateurs sur les rôles pour exprimer les restrictions ou contraintes. Un de ces opérateurs, FILLS, spécifie qu'un rôle prend toujours une valeur d'individu spécifique. La figure 2.20 montre le concept EMPLOYÉ qui est défini comme une personne qui travaille pour une organisation, et le concept EMPLOYÉ-UNU qui est défini comme un employé qui travaille pour l'UNU.

```
EMPLOYÉ ⇔ (AND PERSONNE(ALL travaille-pour ORGANISATION))
EMPLOYÉ-UNU ⇔ (AND EMPLOYÉ(FILLS travaille-pour UNU))
```

FIG. 2.20: Classic - Relation entre catégories et/ou instances.

**Catégorie ou instance.** Classic fait une distinction entre les individus et les concepts et ne supporte pas la notion de métaconcept. Et au dire des auteurs de Classic [9], le système n'est pas fait pour les situations où des individus peuvent être vus comme une classe avec des instances. Il n'est donc pas facilement adaptable pour représenter de la méta-connaissance.



### 2.3.7 Graphes conceptuels

La théorie des graphes conceptuels fournit un formalisme permettant de représenter les objets de l'univers du discours. La connaissance est modélisée par des concepts et des relations conceptuelles. Un concept représente un objet d'intérêt appelé aussi objet de connaissance. Les relations conceptuelles permettent d'associer les concepts. Un graphe conceptuel est un ensemble connexe de concepts et de relations conceptuelles. La théorie des graphes conceptuels a été développée par John Sowa dans les années 80 [70]. Elle constitue un système de logique basé sur les graphes existentiels de C.S. Peirce [62] et sur les réseaux sémantiques. Le lecteur peu familier avec la théorie des graphes conceptuels lira avec intérêt l'annexe (p. 190) qui introduit les principales notions du formalisme des graphes conceptuels.

**Classification et connaissance partielle.** La théorie des graphes conceptuels définit une hiérarchie de types, cette hiérarchie est un treillis avec le type universel  $\top$  au sommet et le type absurde  $\perp$  à la base. L'héritage multiple et la classification multiple sont supportés par le formalisme. L'héritage multiple n'est pas toujours facile à utiliser surtout dans les cas où le nombre de types de concept est très grand. La classification multiple permet les multiples perspectives, chaque perspective ayant son propre vocabulaire.

Dans l'exemple du concessionnaire d'automobiles, l'héritage multiple n'est pas indispensable. Cette situation correspond à plusieurs perspectives : nombre de sièges, mode de propulsion et origine. Et la classification multiple est la plus appropriée. Par exemple, si l'automobile #123 est une 4 portes américaine avec quatre roues motrices, alors nous créons les concepts [4Portes :#123], [4RM :#123], [Amérique :#123], chacun d'eux correspondant à une perspective. Dans le cas d'une connaissance partielle, seuls les concepts correspondant à l'information connue seront créés.

**Relation entre catégories et/ou instances.** Les instances doivent se conformer à la définition du type auquel elles sont associées. Par exemple, la définition du type employé de l'UNU est présentée dans la figure 2.21.

```
Type EMPLOYÉ-UNU(x) is
  [EMPLOYÉ : *x] -> (TRAVAILLE-POUR) -> [ORGANISATION : UNU].
```

FIG. 2.21: Graphes conceptuels - Relation entre catégories et/ou instances.

**Catégorie ou instance.** Un concept est l'association de deux marqueurs : un marqueur qui représente le type du concept et un marqueur qui représente l'instance. Pour changer la perspective il suffit de changer la position du marqueur d'instance. En le faisant passer de droite à gauche on le transforme en marqueur de type. La figure 2.22 illustre comment un individu peut être vu comme un type ou une instance.

```
[CONCEPT-TYPE :ORGANISATION-FÉDÉRALE]-
(SUBTYPE)->[CONCEPT-TYPE :ORGANISATION]
(SYMB)<-[CATEGORY-LABEL :'organisation fédérale']
(DEFINED-BY)<-[GRAPH : Type ORGANISATION-FÉDÉRALE(x) is
                [ORGANISATION :*x]-
                (SYMB)<-[LIBELLÉ :*]
                (ATTR)<-[GOUVERNEMENT :*]
                (MEMBER)<-[ORGANISATION-FÉDÉRALE :*] ].
[ORGANISATION-FÉDÉRALE :AGENCE-ENV]-
(SYMB)<-[LIBELLÉ :'agence environnement']
(ATTR)<-[GOUVERNEMENT :'Canada']
(MEMBER)<-[ORGANISATION-FÉDÉRALE :AGENCE-AIR].
```

FIG. 2.22: Graphes conceptuels : Catégorie ou instance.

Dans [CONCEPT-TYPE :ORGANISATION-FÉDÉRALE] le marqueur ORGANISATION-FÉDÉRALE est à droite et il représente une instance de CONCEPT-TYPE. Lorsque le marqueur est à gauche comme dans [ORGANISATION-FÉDÉRALE :AGENCE-ENV], il représente une catégorie.

### 2.3.8 Synthèse

La table 2.1 présente une synthèse des résultats de la comparaison des différents formalismes de structure.

## 2.4 Formalismes de dynamique

Cette section présente les principaux formalismes permettant de représenter le comportement. Ces formalismes proposent une représentation des processus dans le but de les décrire, de les expliquer, de les échanger ou de les exécuter. Préalablement à cette présentation, nous introduisons les notions de base relatives à la représentation de la dynamique et nous identifions les besoins spécifiques pour la mémoire

	Classification et connaissance partielle	Relation entre catégories et/ou instances	Catégorie ou instance
Entité-Relation-Étendu	Non	Non	Non
Orienté Objet	Oui	Non	même libellé
Modèle relationnel	Vues	Oui	Non
Classic	Oui	Oui	Non
Graphes conceptuels	Oui	Oui	Oui

TAB. 2.1: Synthèse des formalismes de structure.

d'entreprise. Ces besoins vont nous permettre d'évaluer les formalismes. Nous avons retenu quatre formalismes, UML [7] (Unified Modeling Language) qui intègre les Statecharts de Harel, PIF [46] (Process Interchange Format), WfMC [42] (Workflow Management Coalition) et les graphes conceptuels. Les réseaux de Petri [76] purs (sans extensions), souvent utilisés dans les simulateurs, sont peu adaptés à la représentation des processus d'affaires et n'ont pas été retenus dans cette étude.

#### 2.4.1 Notions de base

Les principaux éléments utilisés pour la représentation de la dynamique [72, 68] sont les processus et activités, les participants (intranant, extrant, agent), les événements (pré-condition, post-condition) et les notions de séquence et de parallélisme pour l'ordre d'exécution des activités. Un *processus* est communément vu comme un ensemble d'activités. Une *activité* est une transformation d'entités entrantes en entités sortantes par des agents. Un *événement* marque la fin d'une activité ; l'événement correspond à la réalisation de la post-condition d'une activité et à la réalisation de la pré-condition de l'activité suivante. Un *acteur* est une ressource, humaine ou matérielle, qui rend possible l'exécution d'une activité. Un *intranant* ou *extrant* est une ressource qui est respectivement utilisée ou émise par l'activité. Enfin les notions de séquence et parallélisme définissent l'ordre d'exécution possible des activités. La *séquence* spécifie une dépendance linéaire dans l'exécution des activités tandis que le *parallélisme* spécifie une indépendance d'exécution.

La figure 2.23 illustre les concept d'activité, d'acteur, d'intranant et d'extrant. L'activité est représentée par un cercle, les participants à l'activité sont représentés par des rectangles aux coins arrondis. Les flux de données sont représentés par des flèches

pointillées, une flèche dirigée d'un participant vers l'activité indique un intrant, une flèche dirigée de l'activité vers le participant indique un extrant. Cette représentation simple, indépendante des formalismes étudiés, est utilisée pour présenter les notions de base et les besoins.

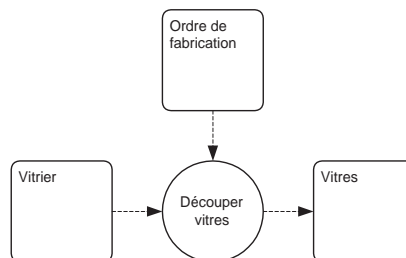


FIG. 2.23: Une activité avec ses participants.

La figure 2.24 illustre les notions de séquence et parallélisme dans un exemple de processus composé de cinq activités. L'activité **Établir ordre de fabrication** est la première activité du processus, les activités **Fabriquer châssis** et **Découper vitres** se déroulent en parallèle, l'activité **Assembler fenêtre** succède aux activités **Fabriquer châssis** et **Découper vitres**, et enfin l'activité **Livrer fenêtre** suit l'activité **Assembler fenêtre** et termine le processus. Nous ne considérerons que la représentation de processus parallèles ou en séquence, tous les autres cas possibles pouvant se ramener théoriquement à ces deux cas (en décomposant les activités en sous-activités).

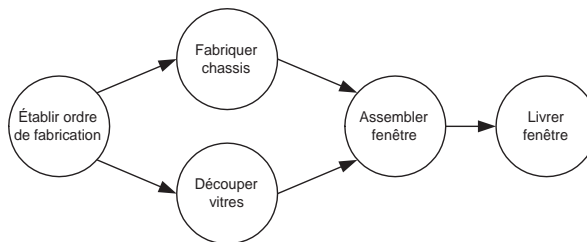


FIG. 2.24: Un processus comme une suite d'activités.

### 2.4.2 Les besoins

Cette section présente les deux principaux besoins de représentation. Il est clair qu'il y a d'autres besoins pour la représentation de la partie dynamique d'une mémoire d'entreprise. Cependant les deux besoins présentés ici sont particuliers et discriminants pour l'évaluation des formalismes. Ces deux besoins sont (i) la représentation de processus partageant des mêmes activités et (ii) le problème de la gestion des instances qui interviennent dans un processus.

#### Le partage d'activités

Considérons le cas de deux processus qui ont une activité en commun. La figure 2.25 présente un exemple de partage d'activités entre deux processus.

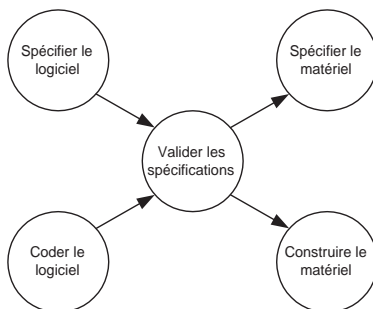


FIG. 2.25: Le partage d'activités.

Il s'agit de l'exemple de la fabrication d'un produit qui est composé d'une partie matérielle et d'une partie logicielle comme par exemple un téléphone cellulaire, un four à micro-onde ou encore un ordinateur. Le premier processus décrit le développement de la partie logicielle. Il est composé des activités **Spécifier le logiciel**, **Valider les spécifications** et **Coder le logiciel**. Le deuxième processus décrit le développement de la partie matérielle. Il est composé des activités **Spécifier le matériel**, **Valider les spécifications** et **Construire le matériel**. L'activité **Valider les spécifications** est une activité de synchronisation et de validation et est commune aux deux processus. La difficulté est la représentation et l'identification des deux processus, celui concernant le logiciel et celui concernant le matériel.

*Pour supporter les besoins de représentation de la dynamique, le formalisme doit supporter la représentation de processus partageant une même activité.*

## Gestion des instances

Pour comprendre le problème de gestion des instances, nous prendrons l'exemple d'une fabrique de fenêtres qui a un service de fabrication pour les fenêtres de taille non standard. La figure 2.26 illustre cet exemple.

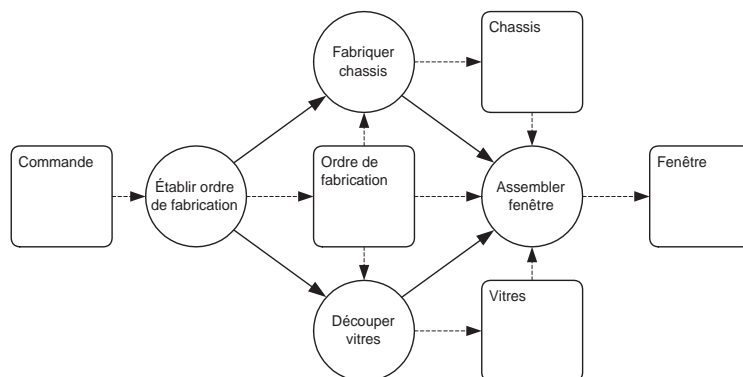


FIG. 2.26: Le problème de la fenêtre.

À partir d'une commande, un ordre de fabrication est établi. Il définit la taille et le matériau pour le châssis et la taille et l'épaisseur des vitres à y insérer. Cet ordre de fabrication est transmis à l'équipe qui fabrique les châssis et à l'équipe qui découpe les vitres. Ensuite le châssis et les vitres sont transmis à l'équipe d'assemblage qui pose les vitres dans les châssis. Le problème est de poser les bonnes vitres (taille et épaisseur) dans les bons châssis. En effet, certains châssis prennent plus de temps à construire que d'autres, il est donc possible que les châssis sortent dans un ordre différent de celui des vitres correspondantes. La résolution du problème se fait par l'équipe d'assemblage qui s'assure par l'ordre de fabrication qu'elle assemble les bons châssis et vitres en fenêtres.

*Pour supporter les besoins de représentation de la dynamique, le formalisme doit supporter la représentation et la gestion des instances dans les processus.*

### 2.4.3 Unified Modeling Language

Nous avons déjà vu la représentation de la partie structure (statique) de UML [7] (Unified Modeling Language) à la section 2.3.4. Rappelons que UML, développé par

Grady Booch, Jim Rumbaugh et Ivar Jacobson, de l'unification des méthodes Booch, OMT et OOSE, est aujourd'hui considéré comme un standard de facto.

UML offre plusieurs types de diagrammes permettant de montrer les différents aspects de la dynamique. Les *diagrammes "Use Case"* montrent les interrelations entre les fonctions offertes par un système et les acteurs externes qui utilisent ces fonctions. Les *diagrammes de séquence* et les *diagrammes de collaboration* spécifient les interactions entre objets en montrant les messages échangés entre les différents objets. Les diagrammes de séquence mettent en valeur la séquence d'échange de messages entre les objets en fonction du temps, alors que les diagrammes de collaboration mettent en valeur les messages échangés entre les objets. Les deux types de diagrammes véhiculent les mêmes informations mais les présentent différemment. Les *diagrammes d'états* décrivent le comportement des objets d'une classe ou d'une méthode en réponse à des stimuli. Plus précisément, un diagramme d'états montre la séquence des différents états par lesquels passe un objet durant sa vie et spécifie les stimuli qui occasionnent les changements d'état ainsi que les réponses et actions de l'objet à ces stimuli. Les *diagrammes d'activités* servent à décrire des processus impliquant plusieurs types d'objets. Un diagramme d'activité est un cas particulier d'un diagramme d'états où les états représentent l'accomplissement d'activités.

Dans le cas d'une mémoire d'entreprise, les diagrammes d'activités sont les plus pertinents et nous en présentons ci-dessous les concepts principaux. UML fait la distinction entre une activité unitaire (ActionState) qui représente l'exécution atomique d'une action et une activité complexe (ActivityState) qui représente l'exécution d'un processus, c'est-à-dire l'exécution d'une séquence non-atomique. Un flux de données (ObjectFlowState) représente l'échange d'un objet entre deux activités, il correspond à la notion d'intrant et d'extrant. Les acteurs sont le plus souvent représentés par des couloirs (Swimlane) dans les diagrammes d'activités. Cependant il est toujours possible de définir un acteur comme un participant et de le relier explicitement à une activité. La figure 2.27 montre les représentations d'une activité et du processus de fabrication de fenêtre en utilisant le formalisme UML. Les diagrammes d'activités montrent un scénario possible, c'est-à-dire, ils montrent des instances d'objets et non des classes.

**Partage d'activités** Tel que présenté dans [7], UML ne semble pas supporter la représentation du partage d'activités. Cependant l'introduction des diagrammes d'activités est relativement récente et tous les cas d'utilisation n'ont pas été présentés.

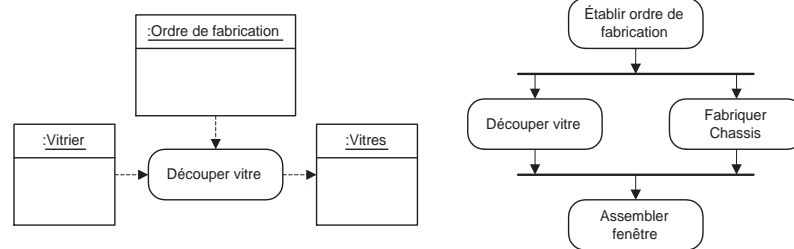


FIG. 2.27: UML - Représentation d'une activité et d'un processus.

**Gestion des instances** La représentation des activités se faisant au niveau instances, le problème de la fenêtre peut être traité. La figure 2.28 montre la représentation UML du problème de la fenêtre. La partie gauche présente un scénario, c'est-à-dire, un exemple du processus avec des objets ; la partie droite présente les relations existantes entre les objets.

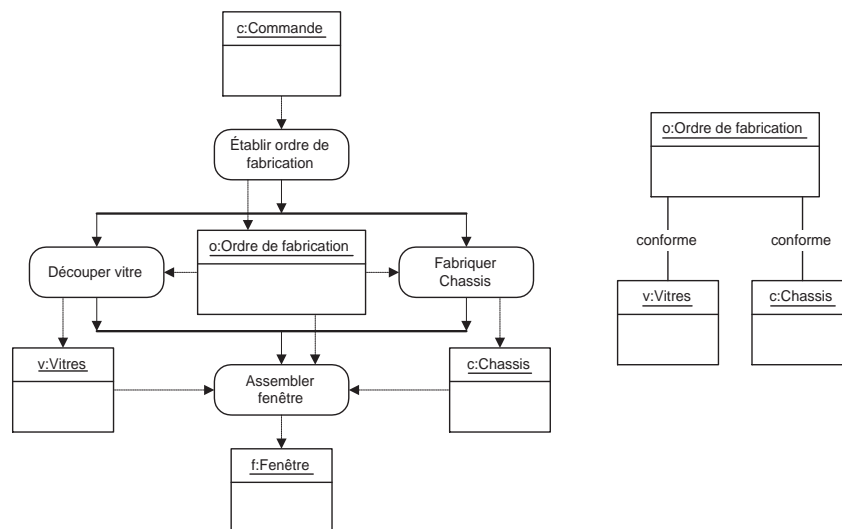


FIG. 2.28: UML - Représentation du problème de la fenêtre.

#### 2.4.4 Process Interchange Format (PIF)

Le groupe de travail PIF (Process Interchange Format), composé de personnes de l'industrie et des universités, a développé un format d'échange et un cadre de référence pour la modélisation des processus [46].



La description d'un processus dans PIF consiste en un ensemble de définitions de frame, chaque frame spécifiant une instance d'une classe du métamodèle. Le métamodèle de PIF (figure 2.29) est composé d'une classe générique ENTITÉ dont toutes les autres classes sont des spécialisations et de quatre classes de base : ACTIVITÉ, OBJET, POINT DANS LE TEMPS, RELATION.

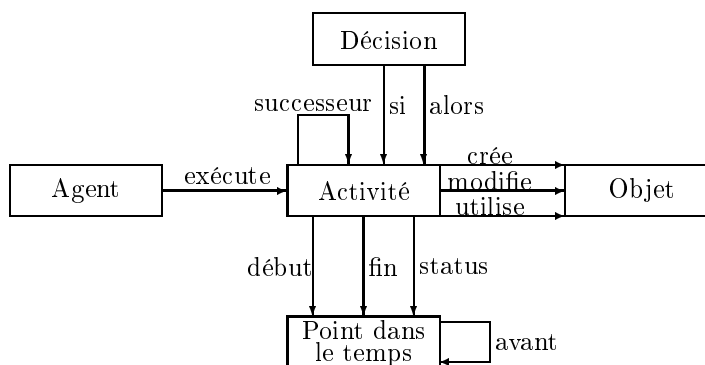


FIG. 2.29: Le métamodèle de PIF.

Les classes ACTIVITÉ et OBJET ont pour sous-classes respectivement DÉCISION et AGENT. La classe relation a sept sous-classes, les sous-classes CRÉE, MODIFIE, EXÉCUTE, et UTILISE qui définissent les relations entre ACTIVITÉ et OBJET, la sous-classe AVANT qui définit la relation de prédécesseur entre deux points dans le temps, la sous-classe SUCCESSEUR qui définit la relation de succession entre deux activités et enfin la sous-classe STATUS qui définit l'état d'une activité à un instant donné.

La description d'une activité est illustrée par l'exemple ci-dessous qui montre la définition de l'activité Découper vitres.

```

(define-frame ACTIVITE                                :own-slots
  :own-slots                                         ((Instance-Of AGENT)
  ((Instance-Of ACTIVITE)                            (Name "Vitrier")))
  (Name "Découper vitres")
  (End FIN-ACTIVITE))                               (define-frame EXECUTE
                                                    :own-slots
                                                    ((Instance-Of PERFORMS)
                                                    (Actor ACTEUR)
                                                    (Activity ACTIVITE)))

(define-frame FIN-ACTIVITE                            (define-frame INTRANT
  :own-slots
  ((Instance-Of TIMEPOINT)))

```

```

:own-slots
((Instance-Of OBJECT)
(Name "Ordre de fabrication"))))

(define-frame UTILISE
:own-slots
((Instance-Of USES)
(Activity ACTIVITE)
(Object INTRANT)))

(define-frame EXTRANT
:own-slots
((Instance-Of OBJECT)
(Name "Vitres")))

(define-frame CREE
:own-slots
((Instance-Of CREATES)
(Activity ACTIVITE)
(Object EXTRANT)))

```

Dans le standard PIF, il n'y a pas de définition explicite pour un processus. Un processus est l'ensemble des activités qui ont été définies. L'exemple ci-dessous montre la représentation d'un processus.

```

(define-frame ACT1
:own-slots
((Instance-Of ACTIVITY)
(Name "Établir ordre de fabrication")
(End FIN-ACT1)))

(define-frame FIN-ACT1
:own-slots
((Instance-Of TIMEPOINT)))

(define-frame ACT2
:own-slots
((Instance-Of ACTIVITY)
(Name "Fabriquer châssis")
(End FIN-ACT2)))

(define-frame FIN-ACT2
:own-slots
((Instance-Of TIMEPOINT)))

(define-frame ACT3
:own-slots
((Instance-Of ACTIVITY)
(Name "Découper vitres")
(End FIN-ACT3)))

(define-frame FIN-ACT3
:own-slots
((Instance-Of TIMEPOINT)))

(define-frame ACT4
:own-slots
((Instance-Of ACTIVITY)
(Name "Assembler fenêtre")
(End FIN-ACT4)))

(define-frame FIN-ACT4
:own-slots
((Instance-Of TIMEPOINT)))

(define-frame ACT5
:own-slots
((Instance-Of ACTIVITY)
(Name "Livrer fenêtre")
(End FIN-ACT5)))

(define-frame FIN-ACT5
:own-slots
((Instance-Of TIMEPOINT)))

(define-frame ACT1-ACT2
:own-slots

```

```

((Instance-Of BEFORE)
(Preceding-Timepoint FIN-ACT1)
(succeeding-Timepoint FIN-ACT2)))

(define-frame ACT1-ACT3
 :own-slots
 ((Instance-Of BEFORE)
 (Preceding-Timepoint FIN-ACT1)
 (succeeding-Timepoint FIN-ACT3)))

(define-frame ACT2-ACT4
 :own-slots
 ((Instance-Of BEFORE)
 (Preceding-Timepoint FIN-ACT2)
 (succeeding-Timepoint FIN-ACT4)))

((Instance-Of BEFORE)
(Preceding-Timepoint FIN-ACT1)
(succeeding-Timepoint FIN-ACT2)))

(define-frame ACT3-ACT4
 :own-slots
 ((Instance-Of BEFORE)
 (Preceding-Timepoint FIN-ACT3)
 (succeeding-Timepoint FIN-ACT4)))

(define-frame ACT4-ACT5
 :own-slots
 ((Instance-Of BEFORE)
 (Preceding-Timepoint FIN-ACT4)
 (succeeding-Timepoint FIN-ACT5)))

```

**Partage d'activités** Le standard PIF permet de représenter les deux séquences d'exécution des activités de développement de logiciel et matériel, comme le montre l'exemple ci-dessous. Cependant il est impossible d'identifier les deux processus autrement que par des commentaires.

```

(define-frame ACT-A
 :own-slots
 ((Instance-Of ACTIVITY)
 (Name "Spécifier le matériel")
 (End FIN-ACT-C)))

(define-frame FIN-ACT-A
 :own-slots
 ((Instance-Of TIMEPOINT)))

(define-frame ACT-B
 :own-slots
 ((Instance-Of ACTIVITY)
 (Name "Valider les spécifications")
 (End FIN-ACT-B)))

(define-frame FIN-ACT-B
 :own-slots
 ((Instance-Of TIMEPOINT)))

(define-frame ACT-C
 :own-slots
 ((Instance-Of ACTIVITY)
 (Name "Construire le matériel")
 (End FIN-ACT-C)))

(define-frame FIN-ACT-C
 :own-slots
 ((Instance-Of TIMEPOINT)))

(define-frame ACT-A1
 :own-slots
 ((Instance-Of ACTIVITY)
 (Name "Spécifier le logiciel")
 (End FIN-ACT-A1)))

(define-frame FIN-ACT-A1
 :own-slots
 ((Instance-Of TIMEPOINT)))

(define-frame ACT-C1
 :own-slots

```

```

((Instance-Of ACTIVITY)
(Name "Coder le logiciel")
(End FIN-ACT-C1)))

(define-frame FIN-ACT-C1
:own-slots
((Instance-Of TIMEPOINT)))

;;; Processus 1 ;;;;
(define-frame ACT-A-ACT-B
:own-slots
((Instance-Of BEFORE)
(Preceding-Timepoint FIN-ACT-A)
(succeeding-Timepoint FIN-ACT-B)))

(define-frame ACT-B-ACT-C
:own-slots

((Instance-Of BEFORE)
(Preceding-Timepoint FIN-ACT-B)
(succeeding-Timepoint FIN-ACT-C1)))

;;; Processus 2 ;;;;
(define-frame ACT-A1-ACT-B
:own-slots
((Instance-Of BEFORE)
(Preceding-Timepoint FIN-ACT-A1)
(succeeding-Timepoint FIN-ACT-B)))

(define-frame ACT-B-ACT-C1
:own-slots
((Instance-Of BEFORE)
(Preceding-Timepoint FIN-ACT-B)
(succeeding-Timepoint FIN-ACT-C1)))

```

**Gestion des instances** Le standard PIF définit les activités et les objets impliqués dans les activités au niveau des types. Le standard n'offre aucun moyen pour contraindre les instances, il n'est donc pas possible de représenter le problème de la fenêtre.

#### 2.4.5 Workflow Reference Model

La Workflow Management Coalition (WfMC) définit dans son modèle de référence [42] un métamodèle de base pour la définition des processus. Le modèle de référence utilise six types d'objets de base pour définir des processus simples. Les types d'objets sont : type de workflow ou processus, activité, rôle, transition, données échangées et application invoquée. La figure 2.30 présente ce métamodèle.

La WfMC a également publié une version bêta [80] d'un format d'échange pour les outils de gestion de processus (Workflow Engine). L'exemple ci-dessous montre la représentation de l'activité *Découper vitres* avec ses participants dans ce format d'échange. Les participants intrants, les ressources exécutant l'activité et les données impliquées dans l'activité sont définies explicitement. Les données qui sont des produits de l'activité sont spécifiées dans les post-conditions de l'activité ou sont définies comme paramètres de sortie des applications invoquées lors de l'exécution de l'activité.

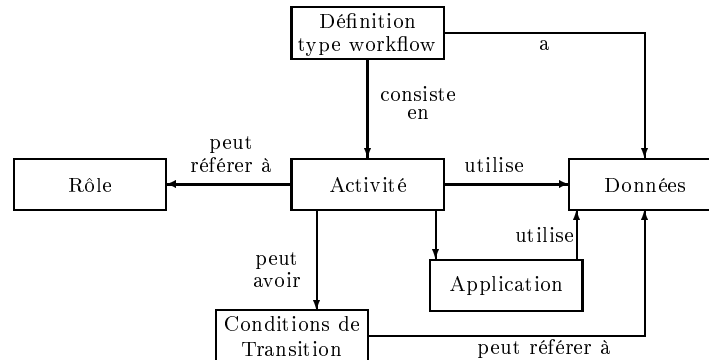


FIG. 2.30: Le métamodèle du WfMC.

```

ACTIVITY Découper_vitres
  PARTICIPANT Vitrier,
    Ordre_de_fabrication
  POST_CONDITION Vitres exists
END_ACTIVITY

DATA Ordre_de_fabrication
  TYPE COMPLEX_DATA
END_DATA

DATA Vitres
  TYPE REFERENCE
END_DATA

PARTICIPANT Vitrier
  TYPE HUMAN
END_PARTICIPANT
  
```

Dans le formalisme de WfMC, un processus est défini par la liste des activités du processus et la liste des transitions qui spécifient l'ordre d'exécution des activités. L'exemple ci-dessous montre la représentation du processus de fabrication de fenêtres.

```

WORKFLOW FABRIQUER_FENETRE
  . . .
  ACTIVITY Établir_Ordre_de_fabrication
  . . .
  END_ACTIVITY
  ACTIVITY Livrer_fenêtre
  . . .
  END_ACTIVITY
  ACTIVITY Découper_vitres
  . . .
  END_ACTIVITY
  TRANSITION
    FROM Établir_Ordre_de_fabrication
    TO Découper_Vitres
  END_TRANSITION
  ACTIVITY Fabriquer_châssis
  . . .
  END_ACTIVITY
  TRANSITION
    FROM Établir_Ordre_de_fabrication
  
```

```

        TO Fabriquer_châssis
END_TRANSITION

TRANSITION
    FROM Découper_Vitres
    TO Assembler_fenêtre
END_TRANSITION

TRANSITION
    FROM Fabriquer_châssis
END_TRANSITION

        TO Assembler_fenêtre
END_TRANSITION

TRANSITION
    FROM Assembler_fenêtre
    TO Livrer_fenêtre
END_TRANSITION

END_WORKFLOW

```

**Partage d'activités** Le formalisme de WfMC définit les processus d'une manière indépendante. Il permet, de plus, de définir les processus impliquant des ressources ou activités communes. L'exemple ci-dessous montre le partage de l'activité Valider les spécifications par les processus 1 et 2. L'activité Valider les spécifications est définie dans le cadre du processus 1 et réutilisée dans la définition du processus 2.

```

WORKFLOW PROCESS1
    ACTIVITY Spécifier_le_logiciel
        . . .
    END_ACTIVITY

    ACTIVITY Valider_les_spécifications
        . . .
    END_ACTIVITY

    ACTIVITY Coder_le_logiciel
        . . .
    END_ACTIVITY

    TRANSITION
        FROM Spécifier_le_logiciel
        TO Valider_les_spécifications
    END_TRANSITION

    TRANSITION
        FROM Valider_les_spécifications
        TO Coder_le_logiciel
    END_TRANSITION

END_WORKFLOW

WORKFLOW PROCESS2
    ACTIVITY Spécifier_le_matériel
        . . .
    END_ACTIVITY

    ACTIVITY Construire_le_matériel
        . . .
    END_ACTIVITY

    TRANSITION
        FROM Spécifier_le_matériel
        TO Valider_les_specifications
    END_TRANSITION

    TRANSITION
        FROM Valider_les_spécifications
        TO Construire_le_matériel
    END_TRANSITION

END_WORKFLOW

```

**Gestion des instances** Le formalisme WfMC définit les modèles de données au niveau des types, cependant les conditions pour déclencher l'exécution (pré-condition) et les conditions qui sont réalisées à la fin de l'exécution d'une activité (post-condition) sont exprimées par des expressions booléennes impliquant des variables. Il est donc possible de représenter les contraintes du processus de fabrication de fenêtre.

#### 2.4.6 Graphes conceptuels et processus

Il n'y a pas de standard défini aujourd'hui pour la représentation des processus. Quelques travaux traitent de la représentation des processus en graphes conceptuels [68, 48, 49]. John Sowa dans [68] traite de la représentation des processus. Les travaux les plus significatifs sont ceux de Dickson Lukose [49] qui a défini un langage de programmation basé sur les graphes conceptuels.

Nous présentons ci-dessous notre proposition de modèle qui répond aux besoins énoncés en 2.4.2. Le modèle comprend les types **ACTIVITE**, **EVENEMENT** et **PROCESSUS**. Une activité est définie comme ayant des intrants, des extrants et des acteurs ; elle dépend de préconditions pour son exécution et sa complétion est marquée par les postconditions. Un événement marque la fin d'une activité et représente le point dans le temps où la postcondition se réalise.

```

TYPE ACTIVITE(x) IS
  [T : ?x]-
    (INTRANT)<-[T]
    (EXTRANT)<-[T]
    (ACTEUR)<-[T]
    (DEPEND-DE)->[PRECONDITION]
    (REALISE)->[POSTCONDITION].

```

```

TYPE EVENEMENT(x) IS
  [T : ?x]-
    (FIN)->[EVENEMENT]
    (SUIT)<-[EVENEMENT].

```

```

TYPE PROCESSUS(x) IS
  [T : ?x]<-(PREMIER)<-[EVENEMENT].

```

En utilisant ce modèle l'activité **DECOUPER-VITRE** est spécifiée par le graphe de définition ci-dessous où les variables de même nom référencent les mêmes objets.

```

TYPE DECOUPER-VITRE(x) IS
  [ACTIVITE : ?x]-

```

```

(ACTEUR)<-[VITRIER]
(INTRANT)<-[ORDRE : *o]
(EXTRANT)<-[VITRE : *v]
(RÉALISE)->[POSTCONDITION : [ORDRE : ?o]->(CONFORME)<-[VITRE : ?v]].

```

Et le processus complet de fabrication d'une fenêtre est représenté par le graphe de définition de type suivant :

```

TYPE FABRIQUER-FENETRE(x) IS
  [PROCESSUS : ?x]-
    (PREMIER)<-[EVENEMENT : *ev1]-
      (FIN)->[ETABLIR-ORDRE-FABRICATION]
      (SUIT)<-[EVENEMENT : *ev2a]-
        (FIN)->[FABRIQUER-CHASSIS] ,
      (SUIT)<-[EVENEMENT : *ev2b]-
        (FIN)->[DECOUPER-VITRE]
        (SUIT)<-[EVENEMENT : *ev3]-
          (SUIT)->[EVENEMENT : *ev2a]
          (FIN)->[ASSEMBLER-FENETRE]
          (SUIT)<-[EVENEMENT : *ev4]
          (FIN)->[LIVRER-FENETRE] .

```

**Partage d'activités** Le modèle processus-activité-événement permet de représenter des processus partageant une même activité. L'exemple ci-dessous présente le partage de l'activité Valider les spécifications par les processus 1 et 2<sup>1</sup>.

```

TYPE PROCESS1(x) IS
  [PROCESSUS : ?x]-
    (PREMIER)<-[EVENEMENT : *ev1a]-
      (FIN)->[SPECIFIER-MATERIEL]
      (SUIT)<-[EVENEMENT : *ev2a]-
        (FIN)->[VALIDER-SPECIFICATION : *vs] ,
        (SUIT)<-[EVENEMENT : *ev3a]-
          (FIN)->[CONSTRUIRE-MATERIEL] .

```

```

TYPE PROCESS2(x) IS
  [PROCESSUS : ?x]-
    (PREMIER)<-[EVENEMENT : *ev1b]-
      (FIN)->[SPECIFIER-LOGICIEL]
      (SUIT)<-[EVENEMENT : *ev2b]-

```

---

<sup>1</sup> En assumant une coréférence globale, c'est-à-dire que des variables de même nom réfèrent le même objet, et ce, pour l'ensemble des graphes du système.



```
(FIN)->[VALIDER-SPECIFICATION : *vs] ,
(SUIT)<-[EVENEMENT : *ev3b]-
(FIN)->[CONSTRUIRE-LOGICIEL] .
```

**Gestion des instances.** Les graphes conceptuels permettent de représenter le problème de la fenêtre. Les définitions des activités ETABLIR-ORDRE, FABRIQUER-CHASSIS, DECOUPER-VITRE, et ASSEMBLER-FENETRE permettent de spécifier que la vitre et le châssis impliqués dans la construction de la fenêtre correspondent au même ordre de fabrication.

```
TYPE ETABLIR-ORDRE(x) IS
  [ACTIVITE : ?x]-
    (INTRANT)<-[COMMANDE]
    (EXTRANT)<-[ORDRE : *o] .

TYPE FABRIQUER-CHASSIS(x) IS
  [ACTIVITE : ?x]-
    (INTRANT)<-[ORDRE : *o]
    (EXTRANT)<-[CHASSIS : *c]
    (REALISE)->[POSTCONDITION : [ORDRE : ?o]<-(CONFORME)<-[CHASSIS : ?c]] .

TYPE DECOUPER-VITRE(x) IS
  [ACTIVITE : ?x]-
    (INTRANT)<-[ORDRE : *o]
    (EXTRANT)<-[VITRE : *v]
    (REALISE)->[POSTCONDITION : [ORDRE : ?o]<-(CONFORME)<-[VITRE : ?v]] .

TYPE ASSEMBLER-FENETRE(x) IS
  [ACTIVITE : ?x]-
    (INTRANT)<-[ORDRE : *o]
    (INTRANT)<-[VITRE : *v]
    (INTRANT)<-[CHASSIS : *c]
    (DEPEND-DE)->[PRECONDITION : [ORDRE : ?o]-
      (CONFORME)<-[VITRE : ?v]
      (CONFORME)<-[CHASSIS : ?c]]
    (EXTRANT)<-[FENETRE] .
```

L'activité ETABLIR-ORDRE établit un ordre de fabrication qui spécifie la vitre et le châssis conformément à la commande. Les deux activités FABRIQUER-CHASSIS et DECOUPER-VITRE ont pour extrants respectivement un châssis et une vitre qui sont conformes à la spécification de l'ordre de fabrication. Enfin l'activité permet d'as-

sembler une vitre et un châssis à condition qu'ils soient conformes au même ordre de fabrication. Le système de gestion de graphes conceptuels doit assurer la création des instances de telle sorte que les coréférences soient respectées.

### 2.4.7 Synthèse

La table 2.2 présente une synthèse des résultats de la comparaison des différents formalismes permettant de représenter la dynamique.

	Partage d'activités	Gestion des instances
UML	Non	Oui
PIF	Oui	Non
WfMC	Oui	Oui
GC	Oui	Oui

TAB. 2.2: Synthèse des formalismes de dynamique.

Les formalismes étudiés ont des caractéristiques très différentes. UML est relativement nouveau et introduit beaucoup de concepts pas toujours bien intégrés. Le nombre élevé de diagrammes pour la représentation des processus dans UML rend son usage difficile. Les standards PIF et WfMC définissent un vocabulaire strict et non flexible. Les graphes conceptuels sont très puissants et flexibles.

## 2.5 Conclusion

Nous avons passé en revue dans ce chapitre les principaux formalismes pour la représentation de la structure des connaissances et la représentation de la dynamique. Aucun des formalismes ne propose une couverture complète des besoins associés à la modélisation et la métamodélisation d'une mémoire d'entreprise. Cependant nous avons vu que le formalisme des graphes conceptuels, même s'il ne fournit pas dans le standard toutes les réponses aux problèmes, offre la puissance d'expression et la flexibilité requises. Notre travail de recherche a consisté à proposer des extensions et à intégrer à ce formalisme les ontologies nécessaires à la représentation des différents types de connaissances nécessaires à la modélisation et la métamodélisation d'une mémoire d'entreprise. Les chapitres suivants présentent notre proposition, résultat de nos travaux.

# Chapitre 3

## Le langage

Dans la littérature, il y a un consensus [63, 41] sur la valeur de la connaissance corporative et sur le fait que sa gestion passe par la mémoire d'entreprise. Or, la construction d'un système de gestion de mémoire d'entreprise exige un modèle de représentation uniforme et non un ensemble de modèles avec des problèmes de compatibilité, de traduction et de pertes d'information dans les échanges entre modèles. Nous nous sommes donc fixés comme but de définir un modèle uniforme répondant à l'ensemble des besoins d'une mémoire d'entreprise.

Nous présentons dans ce chapitre l'architecture de ce modèle uniforme et détaillons le langage qui est la partie fondamentale sur lequel il repose.

### 3.1 Le langage et la pyramide des modèles

Le modèle uniforme s'inscrit dans une architecture de métamodélisation à quatre niveaux. Cette architecture à quatre niveaux a été adoptée par l'OMG [40], UML [7] et CDIF [18].

Les quatre niveaux sont :

- le méta-métamodèle qui décrit les éléments d'un langage de modélisation,
- le métamodèle qui est le langage de modélisation utilisé pour définir les ontologies,
- le modèle qui définit l'ontologie utilisée pour décrire l'application,
- les données qui sont les éléments de l'application.

Ces quatre niveaux peuvent être illustrés par la figure 3.1. Cette pyramide montre l'architecture du modèle que nous nous proposons de développer. Notre travail porte sur la définition du langage permettant de définir les différents niveaux et sur les deux étages supérieurs de la pyramide.

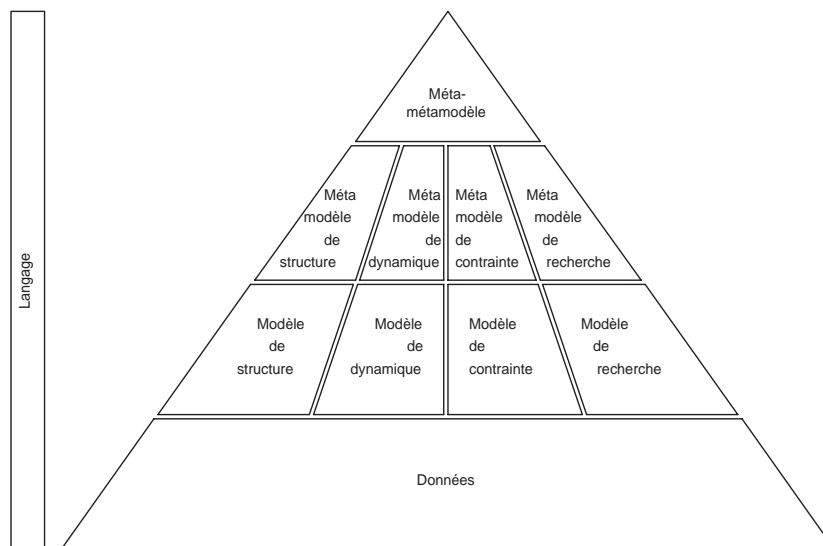


FIG. 3.1: Pyramide des modèles.

Le langage est l'élément fondamental de notre modèle. Le langage est orthogonal à la pyramide des modèles car il est utilisé à tous les niveaux pour représenter la connaissance. Le méta-métamodèle est le niveau le plus élevé et le plus abstrait, il spécifie les objets de définition des métamodèles. Les quatre métamodèles : le métamodèle de structure pour les aspects statiques, le métamodèle de dynamique pour les aspects dynamiques tels les aspects comportementaux et, associés aux deux précédents, le métamodèle de contrainte et le métamodèle de recherche qui permet l'interrogation du contenu, définissent les concepts qui vont permettre la représentation des connaissances corporatives. Les deux niveaux inférieurs, les modèles et les données, correspondent aux niveaux d'application ; ils sont propres à une entreprise, i.e., à un contexte d'application.

Le langage définit les éléments du modèle uniforme utilisés pour décrire tous les niveaux de connaissance. Ces éléments sont décrits en utilisant le modèle uniforme lui-même, c'est-à-dire le langage. Nous présentons ci-dessous brièvement les conventions utilisées.

### 3.1.1 Notation et description du modèle uniforme

Les éléments de base du modèle sont les types de concepts. Ils sont définis par trois catégories de graphes :

- graphe de définition
- graphe de restriction
- graphe de règle

comme illustré dans la figure 3.2. Cette figure présente le graphe de spécification du type de concept *Conducteur*. Les graphes de définition, de restriction et de règles donnent les conditions nécessaires et suffisantes pour la reconnaissance des instances du type de concept *Conducteur*.

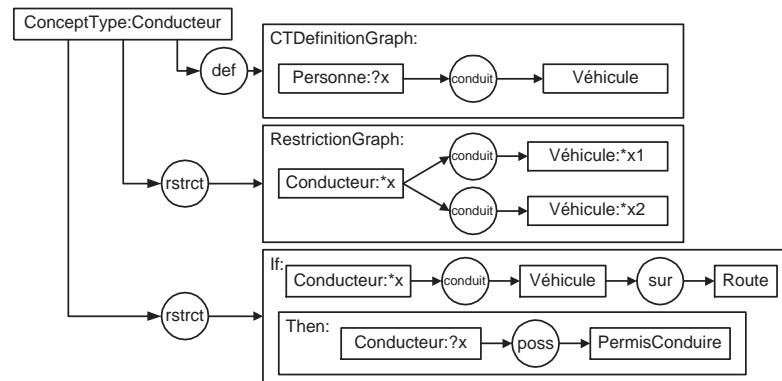


FIG. 3.2: Définition du type *Conducteur*.

**Graphe de définition.** Le graphe de définition présente les relations conceptuelles nécessaires pour qu'un concept soit du type défini. Dans l'exemple un conducteur est une personne qui conduit un véhicule.

D'une manière formelle [70], un type de graphe est défini par une lambda expression, l'exemple de la figure 3.2 correspond à l'équation suivante :

$$\text{Conducteur} = [\text{Personne} : \lambda] \rightarrow (\text{conduit}) \rightarrow [\text{Vehicule}].$$

Le caractère  $\lambda$  indique que le concept *Personne* est le paramètre formel. Dans la version graphique ou la forme linéaire le caractère  $\lambda$  est remplacé par un point d'interrogation.

**Grphe de restriction.** Le ou les graphes de restriction spécifient des conditions nécessaires complémentaires ; ces graphes précisent les relations conceptuelles qui ne peuvent pas exister autour d'un concept. Dans l'exemple, le graphe de restriction précise qu'un conducteur ne peut pas conduire deux véhicules<sup>1</sup>.

**Grphe de règle.** Le ou les graphes de règles spécifient d'autres conditions nécessaires ; ces graphes présentent les règles que doit observer un concept. Dans l'exemple, le graphe de règle exprime le fait que si un conducteur conduit un véhicule sur une route alors il possède un permis de conduire.

### 3.1.2 Exemples

Afin de mieux comprendre les différents niveaux de la figure 3.1, nous allons illustrer par des exemples les types d'information que l'on retrouve à chacun des niveaux. Nous commencerons par illustrer le niveau inférieur, le niveau données, puis nous remonterons dans la pyramide des modèles pour terminer par le langage.

#### Niveau données

Le niveau données représente les objets et les processus d'affaires de l'entreprise. La figure 3.3 montre un graphe conceptuel représentant une occurrence de l'activité *fabriquer un produit*. La fabrication du produit nécessite des matières premières et le résultat est le produit. Le concept [FabriquerProduit : #34] représente l'occurrence de l'activité. Le concept [MatièresPremières : #12] représente l'occurrence des matières premières. Le concept [Produit : #245] représente l'occurrence du produit fabriqué par ce processus utilisant ces matières premières. Les relations utilise (util) et produit (prod) permettent de lier sémantiquement les trois concepts.

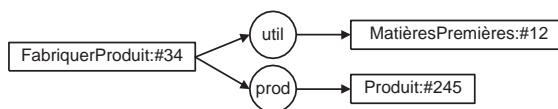


FIG. 3.3: Exemple de graphe conceptuel du niveau données.

Il est à noter que l'exemple ci-dessus sert uniquement d'illustration pour le niveau

<sup>1</sup> Le mécanisme qui s'assure de cet état de chose sera expliqué à la section 3.2.2

données. Une mémoire d'entreprise ne stocke pas les données opérationnelles. Ces données sont gérées par les systèmes transactionnels de l'entreprise (bases de données opérationnelles, systèmes intégrés de gestion, etc.) spécialement adaptés à la gestion des transactions (sécurité, fiabilité et performance).

### Niveau modèle

Le niveau modèle permet la représentation générique des objets et processus d'affaires de l'entreprise. Au niveau modèle sont définis les types Fabriquer Produit, Matières Premières et Produit qui apparaissent dans les concepts du niveau données (figure 3.3). La figure 3.4 montre le graphe conceptuel représentant la spécification de l'activité *fabriquer un produit*. Le graphe décrit le type d'activité *fabriquer un produit*.

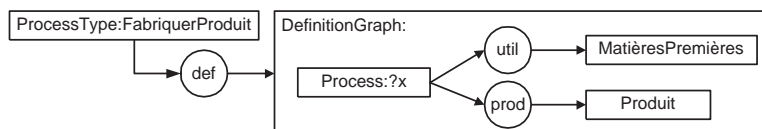


FIG. 3.4: Exemple de graphe conceptuel du niveau modèle.

Ce type est spécifié par un graphe de définition qui montre la forme d'un graphe d'occurrence de l'activité. Le concept [ProcessType : FabriquerProduit] représente le type d'activité. Le concept [DefinitionGraph : . . .] présente le graphe de définition du type d'activité. *Fabriquer un produit* consiste à utiliser des matières premières et produire un produit. Les concepts [Process : ?x], [MatièresPremières] et [Produit] et les relations (util) et (prod) permettent de décrire l'activité d'une manière générique.

### Niveau métamodèle

Le niveau métamodèle présente la spécification des types des concepts apparaissant au niveau modèle. Le niveau métamodèle présente le formalisme utilisé pour décrire les modèles. La figure 3.5 montre le graphe conceptuel spécifiant le métatype ProcessType.

Un ProcessType est un type de concept de modèle (ModelType) qui est spécifié par un graphe de définition qui montre la forme d'une instance du métatype. Une instance du métatype ProcessType est un type de processus, il est défini par un graphe de définition qui montre que ses instances sont des sous-types de Process.

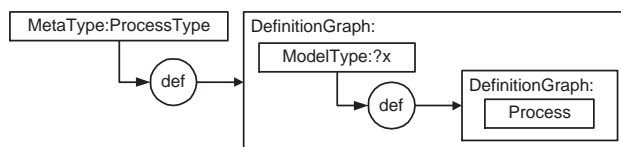


FIG. 3.5: Exemple de graphe conceptuel du niveau métamodèle.

### Niveau métamétamodèle

Le niveau métamétamodèle présente la spécification des types des concepts apparaissant au niveau métamodèle. La figure 3.6 montre le graphe conceptuel définissant le métatype de concept `MetaType`.

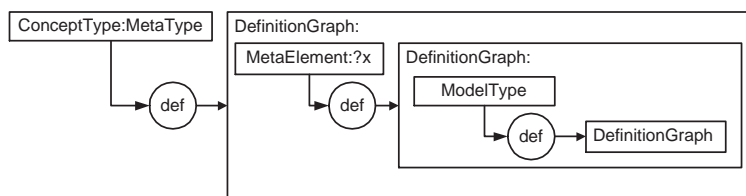


FIG. 3.6: Exemple de graphe conceptuel du niveau métamétamodèle.

`MetaType` est un métatype dont le graphe de définition montre que ses instances sont des types de concepts de modèles (`ModelType`).

### Le langage

Le langage définit les mots qui vont permettre de représenter les différents niveaux de la pyramide des modèles. La figure 3.7 montre les graphes de spécification de `Type`. Les types et relations présentés dans les graphes de spécification seront vus à la section 3.2.

#### 3.1.3 Graphes Conceptuels et métamodélisation

##### État de l'art

Peu de travaux ont été réalisés sur l'utilisation des graphes conceptuels pour la métamodélisation. John Esch [30] effleure le sujet et propose l'introduction de deux relations : `Kind` qui lie un concept au type dont il est instance et `Subt` qui lie deux



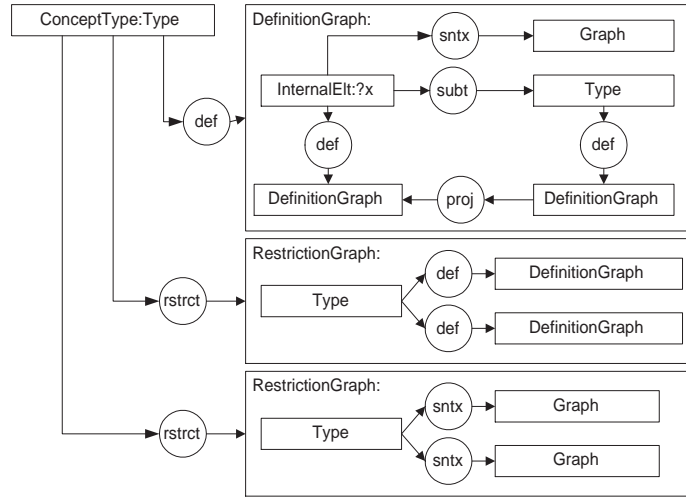


FIG. 3.7: Exemple de graphe conceptuel du langage.

types tels que l'un est sous-type de l'autre. Il définit grâce à la relation *Subt* des hiérarchies de types pour chacun des niveaux et relie les éléments des hiérarchies par la relation *Kind*. Esch n'aborde pas le problème des relations et types de relation ce que nous faisons dans cette thèse. Pour sa part, Michel Wermelinger définit de façon plus formelle les types d'ordre supérieur et propose une traduction vers la logique du premier ordre [75]. Il considère qu'il existe une hiérarchie de types qui contient tous les types et une seule hiérarchie de types de relations qui contient tous les types de relations. Wermelinger classe les types de concepts suivant leur nature et leur ordre. Il distingue les types de concepts relationnels dont les instances sont des types de relations et les autres types de concepts dont les instances sont des concepts ou des types de concepts et ne sont pas des types de relations. Les types de relations sont eux classifiés en fonction de leur arité et de leur ordre. Les instances de types de relations sont des relations.

Ces deux approches sont compatibles et notre travail étend les notions introduites par ces auteurs. Nous définissons deux hiérarchies de types, une pour les types de concepts et une pour les types de relations. Les types de concepts et les types de relations sont classifiés en fonction de leur sémantique. De plus, nous unifions les opérateurs  $\tau$  et  $\rho$  [67] en un seul opérateur  $\omega$  qui permet de passer d'un niveau au niveau inférieur, tel qu'expliqué à la section 3.3.

## Modélisation et lien d'instance

L'activité de modélisation consiste en partie à définir des catégories ou types qui permettent de regrouper des entités ayant des propriétés communes. Le lien entre une entité et le type qui la modélise est communément appelé lien d'instance. Une entité est dite *instance* d'un type.

Spécifier ou déclarer un type, c'est décrire les caractéristiques communes des entités qui vont être instances du type. Les caractéristiques communes sont les attributs des entités, les associations qu'entretiennent les entités, et le comportement des entités. Les attributs décrivent les entités d'un point de vue structurel. Les associations décrivent les entités d'un point de vue fonctionnel. Le comportement décrit les entités d'un point de vue dynamique.

Dans une architecture à quatre niveaux, la relation entre deux niveaux est un lien d'instance. Les entités d'un niveau inférieur sont des instances des types du niveau immédiatement supérieur. Les données (niveau données) sont instances des types (niveau modèle), les types sont instances des méta-types (niveau métamodèle), et les méta-types sont instances des méta-méta-types (niveau méta-métamodèle). Cependant un certain nombre de questions peuvent être posées.

Est-ce que la description des instances dépend du niveau ? Par exemple, la description des types est-elle différente de la description des méta-types ? Les attributs, les relations et le comportement des types et des métatypes sont certainement différents mais la façon de les décrire est-elle différente ? Nous ne pensons pas que la description des attributs, des relations et du comportement soit différente selon le niveau où l'on se place.

Est-ce que les liens d'instance que l'on trouve entre deux niveaux sont identiques ? Nous pensons que selon la nature du métamodèle, le lien d'instance entre le niveau modèle et le niveau données peut différer.

En effet, la définition du lien d'instance est partie intégrante du formalisme de représentation. Si le formalisme utilisé pour la représentation des trois niveaux supérieurs de la pyramide, méta-métamodèle, métamodèle et modèle, est le même que celui défini au niveau métamodèle alors la notion de lien d'instance sera la même à tous les niveaux. Si par contre, le formalisme défini au niveau métamodèle est différent alors la notion d'instance diffèrera entre le niveau modèle et le niveau données. Par exemple, si le formalisme des graphes conceptuels est utilisé pour représenter les trois niveaux supérieurs de la pyramide alors le lien d'instance entre ces trois

niveaux est le lien de conformité défini dans le formalisme des graphes conceptuels. Si le formalisme défini au niveau métamodèle est le formalisme orienté objet alors le lien d'instance entre modèle et données sera la notion de lien d'instance définie dans le modèle objet. Cette notion est sensiblement différente de la notion de conformité qui est son équivalent dans le formalisme des graphes conceptuels.

Pour notre travail, le choix du même formalisme de représentation de la connaissance pour les quatre niveaux nous permet d'utiliser la même notion de lien d'instance pour deux niveaux adjacents. Cette notion de lien d'instance dans le formalisme des graphes conceptuels est défini dans le langage. Le lien d'instance appelé conformité est défini dans la spécification de l'élément Concept (section 3.2.4).

## 3.2 Le langage

Cette section présente le langage utilisé pour décrire notre modèle uniforme. Nous avons présenté à la section 2.3.7 les notions de base des graphes conceptuels. Sowa a défini dans [70] le premier modèle des graphes conceptuels. Depuis de nombreux travaux ont porté sur la formalisation du modèle ainsi que sur ses extensions [13, 60, 61, 75]. Mugnier et Chein [13, 61] ont généralisé le modèle de base [70] en levant certaines restrictions (connexité des graphes, treillis de types). Le langage présenté ici s'appuie sur ces différents travaux et sur le standard [19] en cours de définition. Cependant toutes les utilisations du langage ne sont pas définies explicitement. C'est pourquoi, basés sur la terminologie employée dans [61], nous donnons ci-dessous les principes fondamentaux de notre modèle, ainsi que les opérations de base sur les graphes conceptuels. De plus, nous étendons le standard en enrichissant la manière de définir les types de concepts par l'ajout de contraintes que doivent vérifier les instances.

### 3.2.1 Principes

Nous présentons ici les principes fondamentaux qui régissent le langage.

**Principe 1** *Les graphes ne sont pas nécessairement connexes.*

**Principe 2** *L'ensemble des types de concepts forme un sous-ensemble du treillis engendré par la fermeture des types de concepts par les opérations de conjonction (intersection) et disjonction (union).*

**Principe 3** *Les graphes sont sous forme normale. Deux concepts identiques représentant la même entité, sont systématiquement joints. Deux graphes ou parties de graphes ayant un concept commun sont fusionnés pour n'en faire qu'un seul ce qui implique aussi une étape de simplification où toute relation dupliquée est systématiquement éliminée du nouveau graphe.*

**Principe 4** *L'opération de projection est injective. Si deux concepts sont distincts alors ils se projettent sur des concepts distincts.*

**Principe 5** *Tout concept doit évidemment vérifier le graphe de spécification de son type. De plus, pour qu'une relation conceptuelle puisse être établie entre deux concepts, il faut nécessairement que cette relation apparaisse dans au moins un des graphes de définition des types des concepts concernés.*

**Principe 6** *Seules les propositions réputées vraies sont insérées dans la base de connaissance. Toute proposition identifiée comme fausse, c'est à dire, qui n'est pas conforme aux graphes de spécification des types est rejetée au moment de l'acquisition des connaissances.*

### 3.2.2 Opérations de base sur les graphes conceptuels

Nous présentons ici les opérations et définitions de base [70] pour la dérivation et la structuration des graphes conceptuels. La dérivation permet de créer un nouveau graphe à partir de graphes existants.

Soient  $w$  le graphe dérivé de  $u$  et  $v$  ( $u$  et  $v$  pouvant être identiques). Il y a quatre opérations de dérivation :

**Simplification.** Si deux relations conceptuelles sont identiques, une de deux relations peut être supprimée ainsi que ses arcs.

**Restriction de concept.** Un concept de  $u$  peut être restreint en remplaçant son type par un type plus spécialisé (sous-type) ou en remplaçant un concept générique représentant une entité anonyme par un concept individuel représentant une entité identifiée.

**Restriction de relation.** Le type d'une relation de  $u$  peut être remplacé par un sous-type.

**Jointure.** Si un concept  $c$  de  $u$  est identique à un concept  $d$  de  $v$ ,  $w$  est le graphe résultant de l'union de  $u$  et  $v$  en supprimant  $d$  et en reliant les arcs de  $d$  à  $c$ .

Nous présentons maintenant deux définitions et un théorème qui précisent comment les graphes conceptuels peuvent être structurés.

**Définition 1** Un graphe  $w$  est dit *canoniquement dérivable* d'un ensemble de graphes  $\mathcal{A}$  si une des deux conditions suivantes est remplie :

- $w$  est un élément de  $\mathcal{A}$ ;
- $w$  est un graphe dérivé de graphes eux-mêmes canoniquement dérivables de  $\mathcal{A}$ .

**Définition 2** Si un graphe  $u$  est canoniquement dérivable d'un graphe  $v$ ,  $u$  est dit une *spécialisation* de  $v$  et noté  $u \leq v$ , et  $v$  est dit une *généralisation* de  $u$ .

**Théorème 1** La généralisation définit un ordre partiel sur les graphes conceptuels, appelé *hiérarchie de généralisation*. Pour tout graphe  $u$ ,  $v$  et  $w$ , on a les propriétés suivantes :

- réflexive :  $u \leq u$ ;
- transitive : si  $u \leq v$  et  $v \leq w$  alors  $u \leq w$ ;
- antisymétrique : si  $u \leq v$  et  $v \leq u$  alors  $u = v$ ;
- sous-graphe : si  $v$  est un sous-graphe de  $u$  alors  $u \leq v$ ;
- sous-type : si  $u$  est identique à  $v$  excepté que des concepts de  $v$  ont été restreints à des sous-types dans  $u$  alors  $u \leq v$ ;

Enfin nous présentons l'opération fondamentale des graphes conceptuels qui permet le calcul de la relation de spécialisation. Cette opération est à la base du mécanisme de recherche dans les graphes conceptuels.

**Théorème 2** Pour tous graphes conceptuels  $u$  et  $v$  tels que  $u \leq v$ , il existe une correspondance  $\pi : v \rightarrow u$ , où  $\pi(v)$  est un sous-graphe de  $u$  appelé une *projection* de  $v$  dans  $u$ . L'opération de projection a les propriétés suivantes :

- Pour tout concept  $c$  de  $v$ ,  $\pi(c)$  est un concept de  $\pi(v)$  et le type de  $\pi(c)$  est le même ou est un sous-type du type de  $c$ ;
- Pour toute relation conceptuelle  $r$  de  $v$ ,  $\pi(r)$  est une relation conceptuelle de  $\pi(v)$  et le type de  $\pi(r)$  est le même ou est un sous-type du type de  $r$ . De plus si le  $i^{\text{ème}}$  arc de  $r$  est connecté à un concept  $c$  dans  $v$ , alors le  $i^{\text{ème}}$  arc de  $\pi(r)$  est connecté à  $\pi(c)$  dans  $\pi(v)$ .

Le lecteur intéressé trouvera les démonstrations des théorèmes présentés ci-dessus dans [70].

De plus, selon le principe 4 (page 59), si  $c_1$  et  $c_2$  sont deux concepts distincts de  $v$ , noté  $c_1 \neq c_2$ , alors  $\pi(c_1) \neq \pi(c_2)$ .

### 3.2.3 Vue d'ensemble du langage

Le langage définit les éléments de base qui permettent la représentation de la connaissance. La figure 3.8 illustre la hiérarchie des types que l'on retrouve dans le langage<sup>2</sup>.

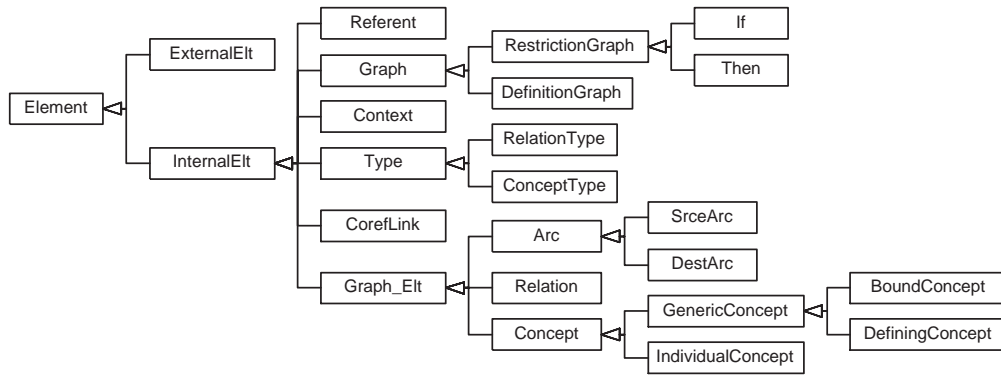


FIG. 3.8: Hiérarchie des types de concepts du langage.

Au plus haut niveau nous distinguons deux types d'élément, les éléments externes (`ExternalElt`), externes au langage et à la pyramide des modèles, et les éléments internes (`InternalElt`) qui sont les constituants du langage. Les éléments externes représentent les objets de l'univers du discours qui sont à l'extérieur du système et qui peuvent être référencés par des éléments internes. Les éléments internes du langage sont regroupés en six types : référent, graphe, contexte, type, lien de co-référence et élément de graphe. Les référents (`Referent`) sont les représentants des objets de l'univers du discours ; les graphes (`Graph`) sont les phrases du langage, les contextes (`Context`) regroupent les graphes représentant la connaissance, les types (`Type`) permettent de regrouper les référents en catégories, les liens de co-référence

<sup>2</sup> Les libellés des type de concept sont en anglais pour des raisons d'intégration avec le standard en cours de définition. Cependant le lecteur trouvera un libellé français de chacun des types de concepts lorsqu'ils seront définis.

(CoRefLink) lie les concepts représentant les mêmes éléments et les éléments de graphes (`Graph_Elt`) sont les éléments, arc (`Arc`), relation (`Relation`) ou concept (`Concept`), que l'on peut trouver dans un graphe.

Parmi les concepts on distingue les concepts individuels (`IndividualConcept`) qui représentent des entités identifiées et les concepts génériques (`GenericConcept`) qui représentent des entités non identifiées.

Parmi les graphes nous distinguons deux sous-types : définition et restriction. Les graphes de définition `DefinitionGraph` sont des graphes utilisés pour la définition des types de concept et des types de relation. Les graphes de restriction `RestrictionGraph` sont des graphes qui doivent être toujours faux et qui contraignent les définitions de types de concept. Les graphes `If` et `Then` sont des cas particuliers de graphes de restriction.

Les figures 3.9 et 3.10 montrent les liens qui existent entre ces types en utilisant le formalisme UML. La première figure présente les éléments qui constituent un graphe. Un graphe est constitué de concepts, de relations et d'arcs source et destination qui lie les concepts et les relations. Un concept est composé de deux parties : un type de concept, lié à d'autres types de concepts par la relation de sous-typage, et un référent qui représente un et un seul élément. Une relation est associée à un type de relation qui définit la sémantique de la relation conceptuelle. Enfin un contexte permet de segmenter l'information en regroupant des graphes par une relation d'extension et en spécifiant par une relation d'intension les graphes pour lesquels les graphes de l'extension sont vrais.

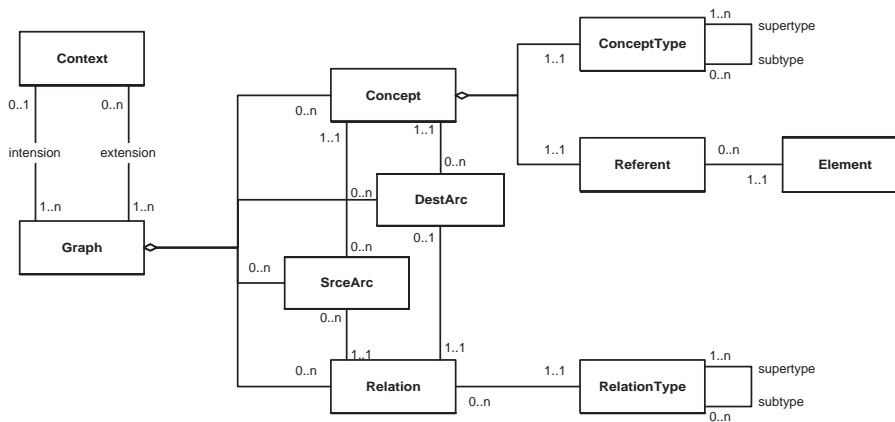


FIG. 3.9: Le modèle du langage - partie 1(formalisme UML).

La figure 3.10 présente les éléments définissant les types, types de concepts et types de relations. Un type est spécifié par : un graphe de définition, des graphes de restriction et des graphes de règles.

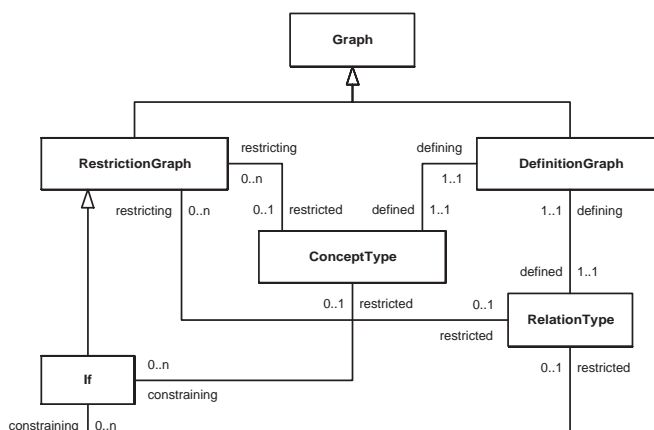


FIG. 3.10: Le modèle du langage - partie 2 (formalisme UML).

La figure 3.11 présente la hiérarchie des types de relations définis dans le langage. La relation de définition `def` et la relation de restriction `rstrct` associent les types à leurs graphes de spécifications. La relation d'appartenance `is-elt` associe les éléments

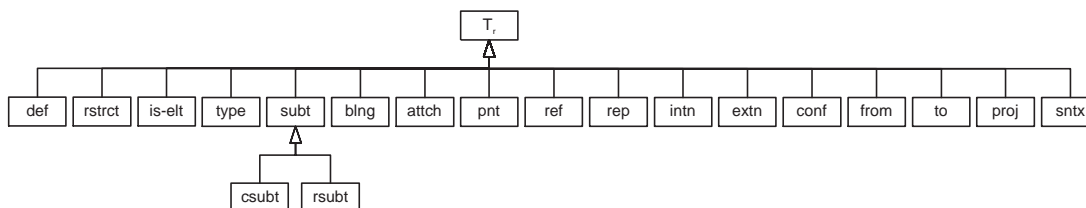


FIG. 3.11: Hiérarchie des types de relations du langage.

d'un graphe au graphe. La relation `type` lie un concept à son type de concept. La relation de sous-typage `subt`, avec ses deux spécialisations `csubt` pour les types de concepts et `rsubt` pour les types de relations, organise les types en hiérarchie. La relation `blng` associe les arcs aux relations et la relation `attach` associe les arcs aux concepts. La relation `pnt` est une relation d'orientation pour les arcs. Les relations `ref` et `rep` associent respectivement un concept à son référent et un référent à l'objet qu'il représente. Les relations `intn` et `extrn` associent un contexte aux graphes qui le



définissent. La relation de conformité `conf` associe un référent à un type. Les relations `from` et `to` associent un lien de co-référence à respectivement la source du lien et la destination du lien. La relation de projection `proj` associe un graphe à un graphe plus spécialisé. Enfin la relation de syntaxe `sntx` associe un type au graphe qui spécifie la syntaxe. Les définitions de ces relations seront données à la section 3.2.5.

### 3.2.4 Les types de concepts du langage

Cette section détaille les types de concept présentés dans la figure 3.8 (page 61). Pour chacun des types nous donnons la définition et le graphe de spécification. Les types sont introduits dans un ordre qui, nous l'espérons, facilite la compréhension des composantes du langage.

#### Grappe conceptuel (Graph)

Basés sur la logique des prédicats du premier ordre, les graphes conceptuels offrent une représentation, graphique ou linéaire, d'un domaine de connaissance.

**Définition 3** *Un graphe conceptuel est un graphe composé de deux sortes de nœuds : des concepts et des relations conceptuelles qui associent les concepts par des arcs orientés.*

La figure 3.12 présente les graphes qui spécifient le type `Graph`. Un graphe conceptuel peut être vide et ne contenir aucun concept, aucune relation conceptuelle et aucun arc. Le graphe de définition est donc réduit à un seul concept.

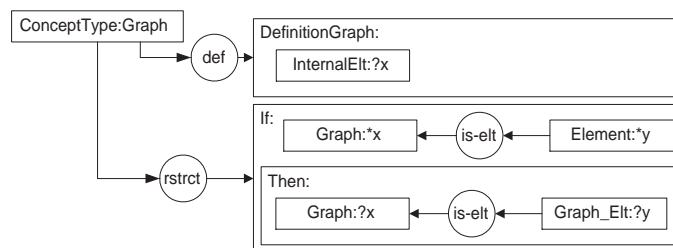


FIG. 3.12: Grappe de spécification de `Graph`.

Le graphe de règles fournit l'information sur les éléments qui peuvent être contenus dans un graphe : si un élément fait partie d'un graphe conceptuel alors cet élément est un `Graph-Elt`, c'est-à-dire, soit un concept, soit une relation, soit un arc. Dans

ce dernier cas, la définition de Arc montre que le graphe contiendrait également un concept et une relation.

Des exemples de graphes sont :

```
[Chat : Garfield] → (mange) → [Lasagne] .
[ConceptType : Personne] → (def) → [DefinitionGraph].
```

Un exemple de concept de type Graph est :

```
[Graph : [Chat : Garfield] → (mange) → [Lasagne] ]
```

### Concept (Concept)

La notion de concept est la notion fondamentale de la théorie des graphes conceptuels. Un concept représente un objet, une idée, ou tout autre notion perceptible et exprimable. Les concepts sont la première sorte de nœuds qui constituent un graphe conceptuel.

**Définition 4** *Un concept est la représentation d'un objet de l'univers du discours. Il est l'assemblage de deux parties : un référent qui identifie l'objet représenté et un type qui classifie l'objet représenté.*

La figure 3.13 présente les graphes qui spécifient le type Concept. Le premier graphe représente la définition suivante : un concept est constitué d'un type, d'un référent et est un élément d'un graphe conceptuel. Les deux graphes de restriction précisent

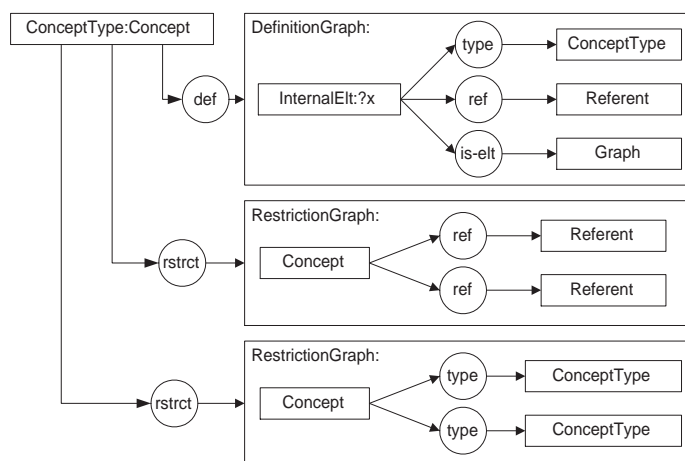


FIG. 3.13: Graphe de spécification de Concept.

les règles de constitution : un concept a un et un seul référent et un et un seul type.

Le référent est dit *conforme* au type. Il est à noter qu'un même type peut être associé à des référents différents et constituer des concepts différents ; de la même façon un référent peut se conformer à des types différents et constituer des concepts différents. Ce mécanisme permet la représentation de points de vue : un même objet physique peut être perçu de manières différentes.

Des exemples de concepts sont :

[Personne], [ConducteurTaxi], [Concept]  
 [Personne : Tom], [Concept : #624], [ConceptType : Personne]

Des exemples de concepts de type Concept sont :

[Concept : [ConceptType : Personne] ], [Concept : [Referent : #624] ]

### Référent (Referent)

Un référent est la partie d'un concept qui représente et identifie l'objet de l'univers du discours dont le concept est une interprétation.

**Définition 5** *Un référent est un représentant d'un objet de l'univers du discours dans la base de connaissance. Un référent est symbolisé par un quantificateur et un 'désignateur' qui réfère à l'objet.*

La figure 3.14 présente le graphe de spécification de Referent. Un référent est un élément interne du langage qui est un constituant d'un concept et qui représente un élément (interne ou externe au langage). Un référent ne peut pas représenter deux éléments et un élément ne peut pas être représenté par deux référents.

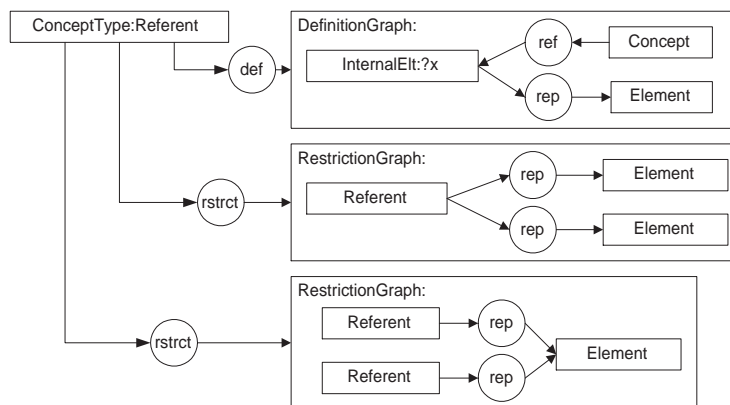


FIG. 3.14: Graphe de spécification de Referent.

Il existe un référent particulier noté #blank qui 'représente' un élément dont l'existence est connue mais dont l'identité est inconnue.

Des exemples de référents sont :

Tom, #12, #blank, Personne.

Des exemples de concepts de type Referent sont :

[Referent : Tom], [Referent : #624], [Referent : Personne]

### Concept générique (GenericConcept)

Il existe deux sortes de concepts suivant que l'élément représenté par le concept est connu ou inconnu.

**Définition 6** *Un concept générique est un concept dont le référent est inconnu. Le concept est une interprétation d'un objet du discours qui existe mais que l'on ne peut pas identifier.*

La figure 3.15 présente le graphe de spécification de GenericConcept. Un concept générique a pour référent #blank qui représente un élément non identifié mais conforme au type. Dans la pratique ce référent est omis.

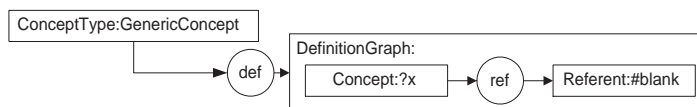


FIG. 3.15: Graphe de spécification de GenericConcept.

Des exemples de concepts génériques sont :

[Personne], [ConducteurTaxi], [Concept]

### Concept individuel (Individual Concept)

**Définition 7** *Un concept individuel est un concept dont le référent est connu. Le concept est une interprétation d'un objet du discours qui est identifié et représenté dans le système par le référent.*

La figure 3.16 présente le graphe de spécification de Individual. Un concept individuel a un référent qui représente un élément. Le référent d'un concept individuel est différent de #blank.

Des exemples de concepts individuels sont :

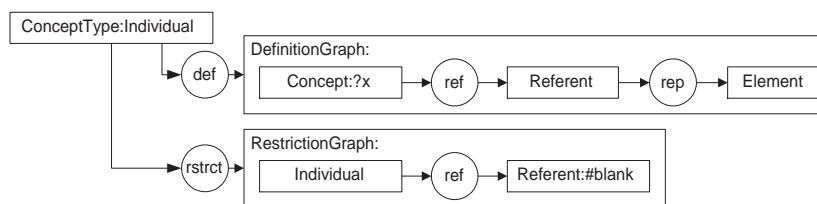


FIG. 3.16: Graphe de spécification de IndividualConcept.

[Personne : Tom], [Referent : #624], [ConceptType : Personne]

**Notation des référents des concepts individuels.** Un référent est noté avec un signe # suivi d'un symbole qui est unique dans le système. Cependant pour simplifier l'écriture, les référents représentant un élément interne du langage ou un élément du système peuvent être remplacés par le symbole de cet élément.

Par exemple, on pourra noter

- [ConceptType : Personne] au lieu de [ConceptType : #812] où #812 est le référent représentant le type Personne.
- [Graph : [Chat : Garfield] → (mange) → [Lasagne] ] au lieu de [Graph : #382] où #382 est le référent représentant le graphe [Chat : Garfield] → (mange) → [Lasagne].
- [String : 'Montréal'] au lieu de [String : #114] où #114 est le référent représentant la chaîne de caractères 'Montréal'.
- [Integer : 2] au lieu de [String : #18] où #18 est le référent représentant le nombre entier 2.

En combinant le référent et l'élément interne référé, on peut définir des éléments nommés. Par exemple, on peut nommer un graphe ou un contexte. On notera [Graph : RepasDeGarfield [Chat : Garfield] → (mange) → [Lasagne] ] pour signifier que le concept [Graph : RepasDeGarfield] représente le graphe [Chat : Garfield] → (mange) → [Lasagne].

Toute référence ultérieure au graphe se fera par [Graph : RepasDeGarfield]. De même on peut écrire [Integer : #18 2] et utiliser indifféremment [Integer : #18] ou [Integer : 2] dans le système puisque ce sont deux représentations de la même entité.

## Lien de co-référence (CoRefLink)

Les liens de co-références permettent d'indiquer que des concepts représentent le même élément, que cet élément soit connu ou non.

**Définition 8** *Un lien de co-référence est un lien orienté entre deux concepts : un concept maître et un concept lié. Le concept lié représente le même élément que le concept maître.*

La figure 3.17 présente le graphe de spécification de CoRefLink. Un lien de co-référence est issu d'un concept, le concept maître, et associé à un autre concept, le concept lié. Un lien de co-référence ne peut pas être issu de deux concepts différents. Il ne peut pas non plus être associé à deux concepts liés différents.

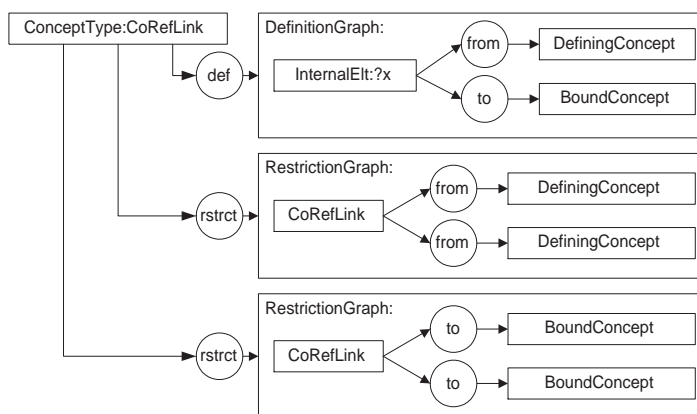


FIG. 3.17: Graphe de spécification de CoRefLink.

Un même concept maître (section 3.2.4) peut être associé à plusieurs concepts liés. Tous les concepts liés représentent alors le même élément, celui défini par le concept maître. L'ensemble de concepts formé par un concept maître et ses concepts liés est appelé un *ensemble de co-référence*. Les liens de co-référence peuvent être représentés par des lignes pointillées ou par des libellés de co-référence dans la partie référent d'un concept. Le libellé de co-référence du concept maître est précédé d'une astérisque et le même libellé de co-référence du ou des concepts liés est précédé d'un point d'interrogation. Tous les concepts d'un ensemble de co-référence ont le même libellé de co-référence.

Des exemples de liens de co-références sont :

[Personne : \*x<sub>1</sub>], [Personne : ?x<sub>1</sub>] et [Chat : \*c Garfield], [Chat : ?c]

### Concept maître (DefiningConcept)

Un concept maître est un concept qui définit l'élément représenté dans un lien de co-référence.

**Définition 9** *Un concept maître est le concept source dans un lien de co-référence. Il définit l'élément représenté par tous les concepts de l'ensemble de co-référence.*

La figure 3.18 présente le graphe de spécification du type DefiningConcept. Un concept maître est un concept dont est issu un lien de co-référence.

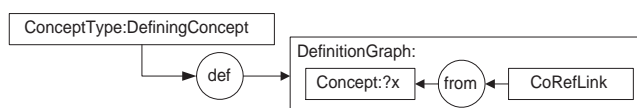


FIG. 3.18: Graphe de spécification de DefiningConcept.

Un concept maître est identifié par un libellé de co-référence précédé d'une astérisque. Un ensemble de co-référence peut être réduit au seul concept maître. L'emploi d'un libellé de co-référence est souvent utilisé dans ce cas pour rappeler visuellement que deux concepts ne sont pas identiques.

Des exemples de concepts maîtres sont :

[Personne : \*x<sub>1</sub>], [Chat : \*c Garfield]

### Concept lié (BoundConcept)

Les concepts liés sont les concepts dont le référent est identifié par le concept maître de l'ensemble de co-référence.

**Définition 10** *Un concept lié est le concept destination dans un lien de co-référence. Son référent est le même que celui du concept maître de l'ensemble de co-référence.*

La figure 3.19 présente le graphe de spécification du type BoundConcept. Un concept lié est un concept destination d'un lien de co-référence.

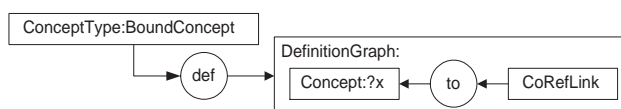


FIG. 3.19: Graphe de spécification de BoundConcept.

Un concept lié est identifié par un libellé de co-référence précédé d'un point d'interrogation.

Des exemples de concepts liés sont :

[Personne : ?x<sub>1</sub>] , [Chat : ?c]

## Arc (Arc)

Les arcs sont les éléments qui permettent de lier les concepts aux relations.

**Définition 11** *Un arc lie une relation à un concept dans un graphe conceptuel. Un arc est dit appartenir à la relation et est dit attaché au concept.*

La figure 3.20 présente le graphe de spécification du type Arc.

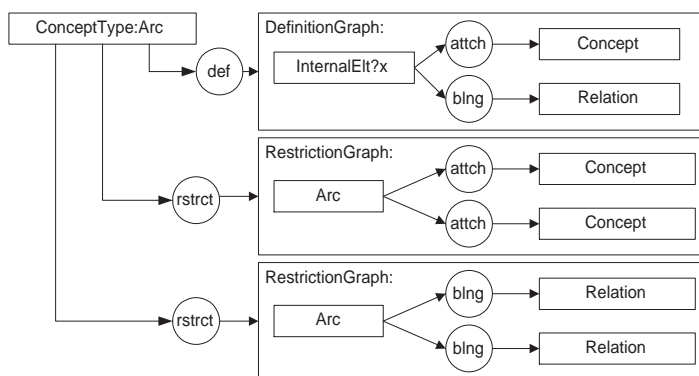


FIG. 3.20: Graphe de spécification de Arc.

Les arcs d'un graphe conceptuel sont orientés, et on distingue les arcs orientés vers la relation, les arcs source, de ceux orientés vers le concept, les arcs destination (voir figure 3.21)

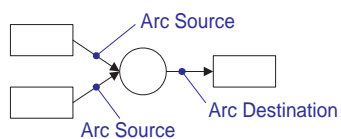


FIG. 3.21: Source et destination.



### Arc Source (SrceArc)

Les arcs source sont les arcs orientés des concepts vers les relations. Une relation conceptuelle peut avoir plusieurs sources.

**Définition 12** *Un arc Source est un arc issu d'un concept et orienté vers la relation.*

La figure 3.22 présente le graphe de spécification du type SrceArc.

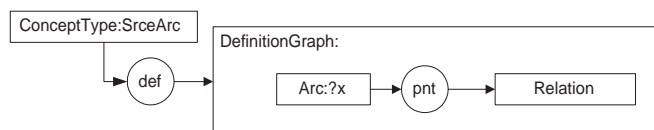


FIG. 3.22: Graphe de spécification de SrceArc.

### Arc Destination (DestArc)

Les arcs destination sont les arcs orientés des relations vers les concepts.

**Définition 13** *Un arc Destination est un arc orienté vers le concept*

La figure 3.23 présente le graphe de spécification du type ArcDest.

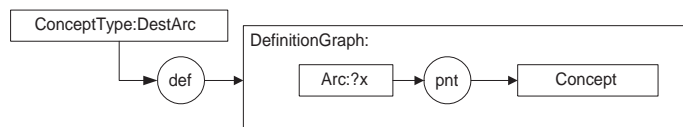


FIG. 3.23: Graphe de spécification de ArcDest.

### Relation Conceptuelle (Relation)

Les relations conceptuelles permettent d'assembler les concepts pour construire une phrase, une idée, une proposition. Les relations conceptuelles sont la deuxième sorte de nœuds qui constituent un graphe conceptuel.

**Définition 14** *Une relation conceptuelle représente une association entre un ou plusieurs concepts. Une relation conceptuelle est composée d'un libellé de relation qui classifie la relation, et d'arcs qui lient la relation aux concepts associés.*

La figure 3.24 présente le graphe de spécification du type Relation. Le graphe de définition montre qu'une relation conceptuelle est associée à un type de relation et est élément d'un graphe conceptuel. Le graphe de restriction précise les cardinalités : une relation ne peut pas être associée à deux types de relation.

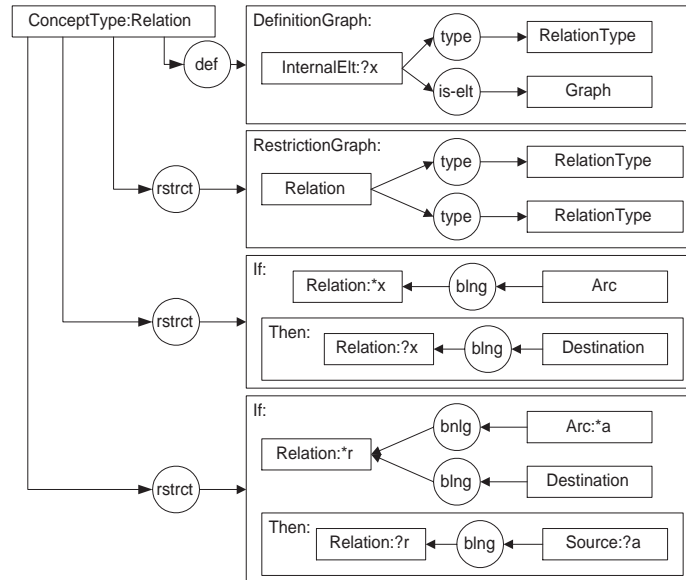


FIG. 3.24: Graphe de spécification de Relation.

Le nombre d'arcs d'une relation conceptuelle est appelé sa *valence*. Parmi les concepts associés, il y a un et un seul concept destination qui est pointé par un arc orienté, et zéro ou plusieurs concepts sources qui sont à l'origine d'un arc orienté (voir figure 3.21).

Pour une relation de valence  $n$ , les  $n - 1$  premiers arcs sont orientés vers la relation (arc source) et le  $n^{\text{ème}}$  est orienté vers un concept (arc destination). Les graphes de règles spécifient cette contrainte. Le premier graphe exprime que s'il existe un arc qui appartient à une relation alors il existe un arc de type destination (possiblement différent) qui appartient à cette relation. Le deuxième graphe exprime que si une relation est liée à un arc destination alors tout autre lien est un arc source.

### Type (Type)

Les types sont les éléments qui permettent de définir et d'organiser d'une manière générique la connaissance. Les types sont définis à partir des propriétés communes

des instances. Les types permettent de regrouper les instances véhiculant une même sémantique. Nous distinguons deux sortes de types : les types de concepts, qui organisent les entités, et les types de relation qui organisent les liens entre entités.

**Définition 15** *Un type représente un ensemble de concepts ou de relations qui ont les mêmes propriétés.*

La figure 3.25 présente le graphe de spécification de Type. Un type est un élément qui a une relation de sous-typage avec un autre type (relation *subt*). Un type est défini (relation *def*) par un graphe de définition qui est nécessairement une projection (relation *proj*) du graphe de définition du supertype telle que les deux concepts liés (ceux ayant comme référent ?x) soient joints et par un graphe qui montre la syntaxe (relation *sntx*) pour les éléments du type. Les graphes de restriction montrent qu'un type a un seul graphe de définition et de syntaxe. Le langage distingue deux sortes de types : les types de concept et les types de relation.

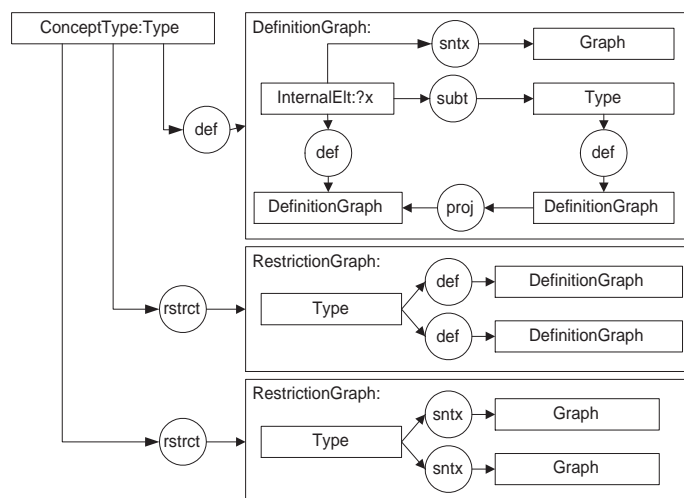


FIG. 3.25: Graphe de spécification de Type.

### Type de concept (ConceptType)

Un type de concept est défini, d'une part par son insertion dans la hiérarchie des types de concepts, et d'autre part par un graphe qui montre les relations que ce type de concept peut avoir.

**Définition 16** *Un type de concept représente un ensemble de concepts ayant les mêmes propriétés, c'est-à-dire les mêmes types de relation avec les mêmes types de concept.*

La figure 3.26 présente la définition d'un type de concept. Un type de concept est un type ayant une relation de sous-typage `csubt` avec un autre type de concept.

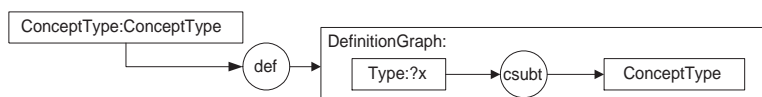


FIG. 3.26: Graphe de spécification de `ConceptType`.

Les types de concepts sont organisés en une hiérarchie de spécialisation par la relation `csubt`. À une extrémité de la hiérarchie il y a le type `Entity` encore noté  $\top_c$  qui représente le type de concept universel dont tous les concepts sont instances et à l'autre extrémité il y a le type `Absurdity` encore noté  $\perp_c$  qui représente le type de concept absurde qui n'a aucune instance.

Des exemples de types de concepts sont :

`[ConceptType : Personne]` , `[ConceptType : Conducteur]` , `[ConceptType : Graph]`

Un exemple de définition de type de concept est présenté à la figure 3.27.

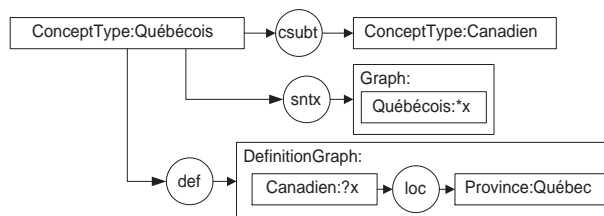


FIG. 3.27: Graphe de la spécification de `Québécois`.

Dans la pratique le graphe présentant la syntaxe et la relation de sous-typage peuvent être omis car ils se déduisent du graphe de définition et la définition du type peut être simplifiée comme illustré par la figure 3.28.

### Type de relation (`RelationType`)

Un type de relation est défini, d'une part par son insertion dans la hiérarchie des types de relation, et d'autre part par un graphe qui montre les types de concepts que

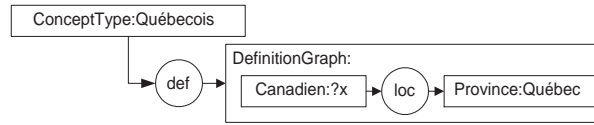


FIG. 3.28: Graphe simplifié de la spécification de Québécois.

ce type de relation peut associer. La figure 3.29 présente la définition d'un type de relation.

**Définition 17** *Un type de relation représente un ensemble de relations qui expriment la même association.*

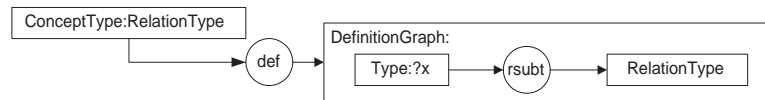


FIG. 3.29: Graphe de spécification de RelationType.

Les types de relations sont organisés en une hiérarchie de spécialisation par la relation  $rsubt$ . À une extrémité de la hiérarchie il y a  $\top_r$  qui représente le type de relation universelle dont toutes les relations sont instances et à l'autre extrémité il y a  $\perp_r$  qui représente le type de relation absurde qui n'a aucune instance.

Des exemples de types de relations sont :

[RelationType :  $subt$ ] , [RelationType :  $mange$ ] , [RelationType :  $def$ ]

Des exemples de définitions de type de relation sont illustrés par les figures 3.30 et 3.31.

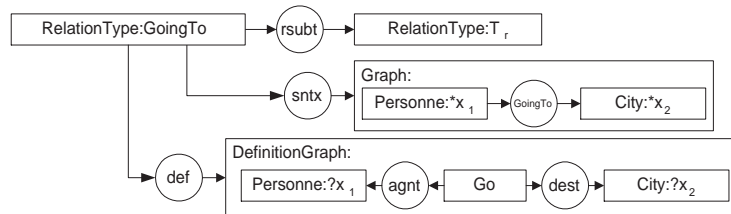


FIG. 3.30: Graphe de la spécification de la relation GoingTo.

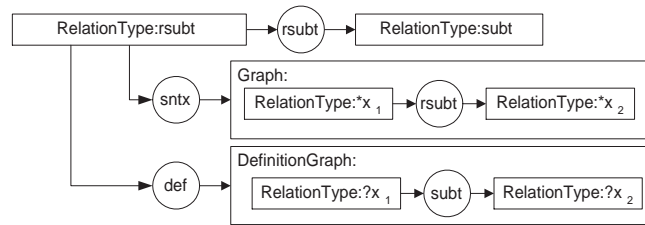


FIG. 3.31: Graphe de la spécification de la relation rsubt.

Dans certains cas le graphe présentant la syntaxe et la relation de sous-typage peuvent être omis car ils se déduisent du graphe de définition et la définition du type peut être simplifiée comme illustré par la figure 3.32.

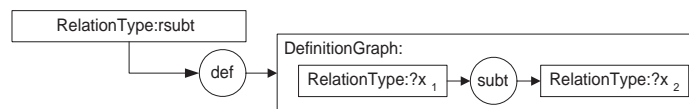


FIG. 3.32: Graphe simplifié de la spécification de rsubt.

### Graphe de définition (DefinitionGraph)

Les graphes de définition permettent de spécifier les types de concepts ou de relations.

**Définition 18** *Un graphe de définition est un graphe qui définit un type en montrant la forme que doit avoir le graphe impliquant les instances de ce type.*

La figure 3.33 présente le graphe de spécification de Definition Graph. Un graphe de définition est un graphe qui définit un type (relation Def). Un même graphe de définition ne peut pas être associé à deux types différents.

### Graphe de restriction (RestrictionGraph)

Les graphes de restriction permettent dans la spécification d'un type de concept de définir les situations qui ne peuvent pas se produire.

**Définition 19** *Un graphe de restriction est un graphe qui spécifie une situation qui n'est pas possible.*

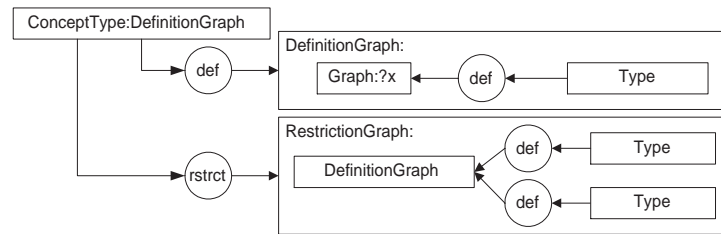


FIG. 3.33: Graphe de spécification de DefinitionGraph.

La figure 3.34 présente le graphe de spécification de Restriction Graph. Un graphe de restriction est un graphe avec une négation. Le graphe doit être toujours faux et donc sa négation vraie.

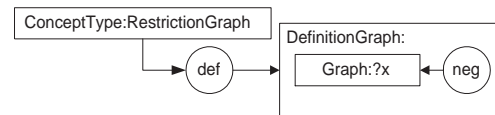


FIG. 3.34: Graphe de spécification de RestrictionGraph.

### Graphe de règle : Si ... (If)

Les graphes de règle If permettent dans la définition d'un type de concept de spécifier les contraintes que doivent vérifier les instances.

**Définition 20** *Un graphe de règles If est un graphe de restriction qui spécifie une règle qui doit être toujours vérifiée.*

Un graphe de règle est composé d'au moins deux sous-graphes. Le premier représente la prémisse, c'est-à-dire, la clause If. Le deuxième représente la conclusion ; il est réduit à un seul concept de type Then. Si le graphe ou les graphes définis dans la clause If sont vrais alors le graphe de la clause Then doit être également vrai.

La figure 3.35 présente le graphe de spécification de If. Un graphe de règle est une restriction, c'est-à-dire, une proposition avec une négation, qui contient une clause Then. Un graphe de règles doit être toujours faux lorsqu'il est évalué.

Montrons que la clause 'Si A alors B' est bien équivalente à deux restrictions imbriquées. Les lignes suivantes sont équivalentes :

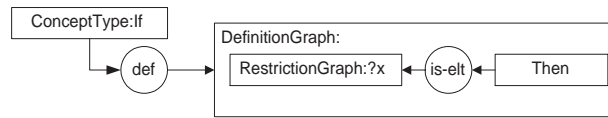


FIG. 3.35: Graphe de spécification de If.

$$\begin{aligned}
 & A \Rightarrow B \\
 & \neg A \vee B \\
 & \neg A \vee \neg\neg B \\
 & \neg (A \wedge \neg B) \\
 & \neg [\text{Graph} : A \neg [\text{Graph} : B] ] \\
 & [\text{RestrictionGraph} : A [\text{RestrictionGraph} : B] ] \\
 & [\text{If} : A [\text{Then} : B] ]
 \end{aligned}$$

### Graphe de règle : ... Alors (Then)

Un graphe de règle Then est la partie 'Alors' d'un graphe de règles.

**Définition 21** *Un graphe de règle Then est une restriction qui apparaît dans un graphe de règles If.*

La figure 3.36 présente le graphe de spécification de Then. Un graphe Then est une proposition avec une négation qui est élément d'un graphe de règle If.

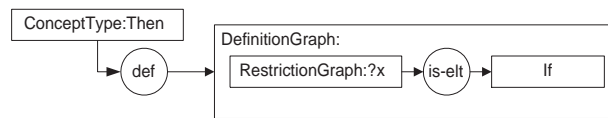


FIG. 3.36: Graphe de spécification de Then.

### Contexte (Context)

La notion de contexte est une notion indispensable pour la modélisation d'une mémoire d'entreprise dans un but de segmentation de l'information. Les contextes permettent de définir des espaces de validité. En effet, une norme peut être applicable dans un département et pas dans un autre ou encore la notion de produit peut différer d'un département à l'autre. Les contextes permettent de modéliser ces situations.



**Définition 22** *Un contexte est composé de deux parties : la première partie, l'intension, est un ensemble de graphes qui décrit les conditions sous lesquelles les graphes de la deuxième partie, l'extension, sont vrais.*

La figure 3.37 présente le graphe de spécification de Context. Un contexte est un élément interne associé par une relation d'intension à un ou plusieurs graphes et par une relation d'extension à un ou plusieurs autres graphes.

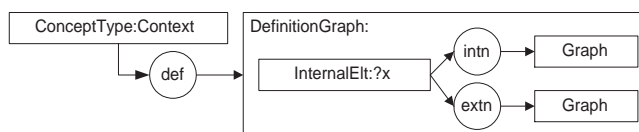


FIG. 3.37: Graphe de spécification de Context.

### 3.2.5 Les relations conceptuelles du langage

Cette section détaille chacune des relations conceptuelles du langage présentées dans la figure 3.11 (page 63). Les relations conceptuelles du langage permettent de structurer les concepts en graphes conceptuels. Ci-dessous nous donnons par ordre alphabétique la syntaxe de chacune des relations conceptuelles du langage.

#### attaché à / attached to (attch)

La relation attch associe un arc au concept auquel il est attaché. La figure 3.38 présente la définition de la relation attch.

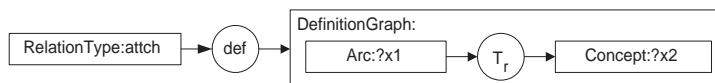


FIG. 3.38: Graphe de spécification de attch.

#### appartient à / belongs to (blng)

La relation blng est une relation d'appartenance. Elle associe un arc à la relation à laquelle il appartient. La figure 3.39 présente la définition de la relation blng.

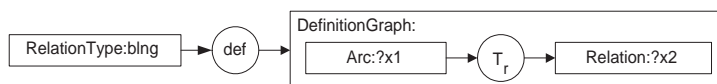


FIG. 3.39: Graphe de spécification de blng.

### conforme / conforms (conf)

La relation de conformité *conf* associe un référent aux types auxquels il se conforme. Un référent  $x1$  est conforme à un type  $x2$  si et seulement si il existe un concept dont  $x1$  est le référent et qui a une relation *type* avec  $x2$ . La figure 3.40 présente la définition de la relation *conf*.

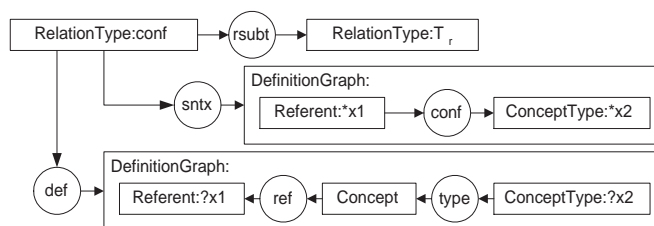


FIG. 3.40: Graphe de spécification de conf.

### sous-type de concept / concept subtype (csubt)

La relation *csubt* est une relation de sous-typage et une spécialisation de la relation *subt*. Elle associe des types de concepts. La figure 3.41 présente la définition de la relation *csubt*.

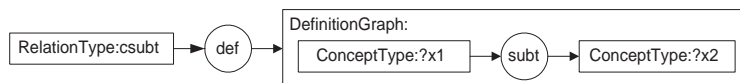


FIG. 3.41: Graphe de spécification de csubt.

### définit / defines (def)

La relation *def* associe un graphe de définition au type de concept qu'il spécifie. La figure 3.42 présente la définition de la relation *def*.

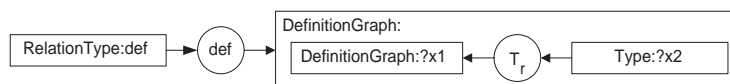


FIG. 3.42: Graphe de spécification de def.

**extension (extn)**

La relation extn associe un contexte à un graphe d'extension. La figure 3.43 présente la définition de la relation extn.

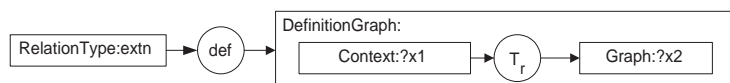


FIG. 3.43: Graphe de spécification de extn.

**de / from (from)**

La relation from associe un lien de co-référence au concept maître qui est la source de la co-référence. La figure 3.44 présente la définition de la relation from.

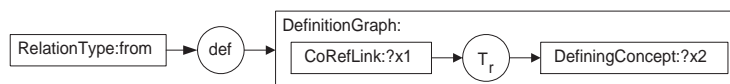


FIG. 3.44: Graphe de spécification de from.

**intension (intn)**

La relation intn associe un contexte à un graphe d'intension. La figure 3.45 présente la définition de la relation intn.

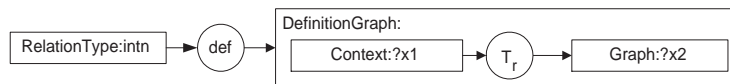


FIG. 3.45: Graphe de spécification de intn.

### est élément de / is element of (is-elt)

La relation is-elt associe un élément interne : concept, relation ou arc au graphe dont il est un élément. La figure 3.46 présente la définition de la relation is-elt.

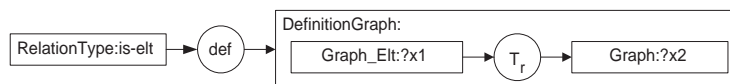


FIG. 3.46: Graphe de spécification de is-elt.

### projection (proj)

La relation proj est une relation de dérivation. Elle associe un graphe à un graphe dérivé. La figure 3.47 présente la définition de la relation proj.

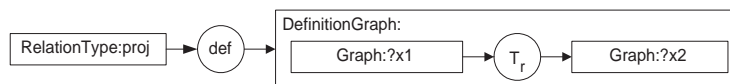


FIG. 3.47: Graphe de spécification de proj.

### pointe vers / points (pnt)

La relation pnt est une relation d'orientation. Elle associe un arc orienté à l'élément vers lequel l'arc pointe. La figure 3.48 présente la définition de la relation pnt.

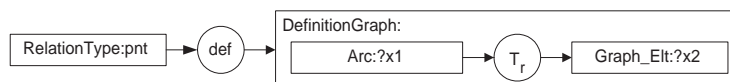
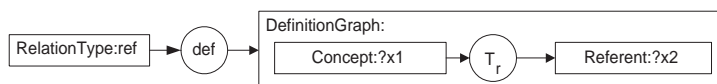


FIG. 3.48: Graphe de spécification de pnt.

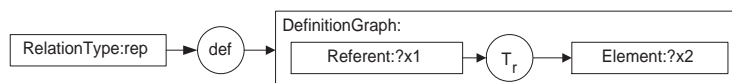
### réfère à / refers to (ref)

La relation ref associe un concept à son référent. La figure 3.49 présente la définition de la relation ref.

FIG. 3.49: Graphe de spécification de *ref*.

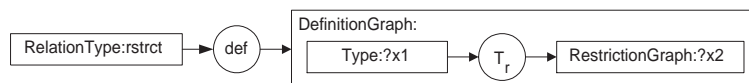
### représente / represents (*rep*)

La relation *rep* associe un référent à l'élément qu'il représente. Le référent est un représentant de l'élément dans le système. La figure 3.50 présente la définition de la relation *rep*.

FIG. 3.50: Graphe de spécification de *rep*.

### restriction (*rstrct*)

La relation *rstrct* associe un type aux graphes de restriction qui contraignent les instances du type. La figure 3.51 présente la définition de la relation *rstrct*.

FIG. 3.51: Graphe de spécification de *rstrct*.

### sous-type de relation / relation subtype (*rsubt*)

La relation *rsubt* est une relation de sous-typage et une spécialisation de la relation *subt*. Elle associe des types de relations. La figure 3.52 présente la définition de la relation *rsubt*.

### syntaxe / syntax (*sntx*)

La relation *sntx* associe un type à son graphe de syntaxe qui montre comment écrire ou représenter les instances du type. La figure 3.53 présente la définition de la

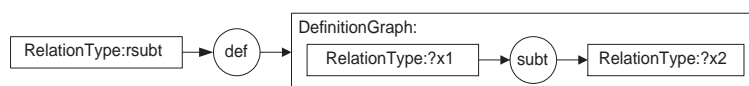


FIG. 3.52: Graphe de spécification de rsbt.

relation sntx.

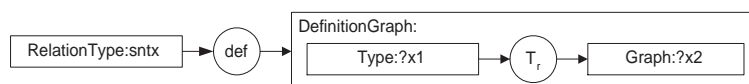


FIG. 3.53: Graphe de spécification de sntx.

### sous-type / subtype (subt)

La relation `subt` est une relation de sous-typage. Elle associe deux types, le premier étant sous-type du deuxième. Le type est plus général et peut être mis en lieu et place du sous-type. La figure 3.54 présente la définition de la relation `subt`.

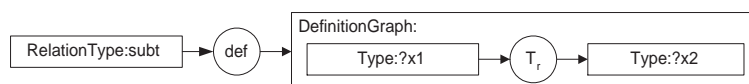


FIG. 3.54: Graphe de spécification de subt.

### type

La relation `type` associe un concept au type dont il est une instance. La figure 3.55 présente la définition de la relation `type`.

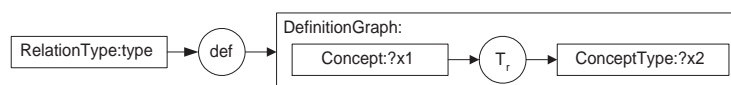


FIG. 3.55: Graphe de spécification de type.

### 3.3 Opérateur de correspondance

Le langage que nous venons de voir permet de spécifier des connaissances à tous les niveaux de la pyramide (figure 3.1 page 51) ainsi que des règles de correspondance entre les niveaux. Dans [67] Sowa décrit le 'mapping' entre le niveau méta et le niveau données. Les graphes conceptuels permettent d'exprimer des assertions aussi bien au niveau méta qu'au niveau données comme dans l'exemple ci-dessous où il est exprimé que le type Entité a pour sous-type Plante.

[Type :Plante]->(subt)->[Type :Entité]. [Plante :  $\forall *x$ ] [Entité : ?x].

Pour traduire une expression du niveau méta (expression de gauche) au niveau données (expression de droite) Sowa définit deux opérateurs notés  $\tau$  et  $\rho$  permettant de passer d'un niveau au niveau inférieur. L'opérateur  $\tau$  traduit le nom du référent en nom de type de concepts comme dans l'exemple ci-dessous qui traduit un énoncé à un niveau en un énoncé au niveau inférieur. Si  $t_1$  est un sous-type de  $t_2$  alors tout référent conforme à  $t_1$  est conforme à  $t_2$ .

IF : [Type : $*t_1$ ]->(subt)->[Type : $*t_2$ ]  
THEN : [ $\tau t_1$  :  $\forall *x$ ] [ $\tau t_2$  : ?x]

L'opérateur  $\rho$  joue le même rôle que  $\tau$  mais pour les types de relations et les relations. L'opérateur transforme le nom d'une relation en un type de relation. L'exemple ci-dessous montre la règle de traduction du niveau méta au niveau données dans un diagramme E-R.


IF : [Type : $*t_1$ ]->(arg1)->[Relation : $*r$ ]->(arg2)->[Type : $*t_2$ ]  
THEN : [ $\tau t_1$ ]->( $\rho r$ )->[ $\tau t_2$ ]

Plus généralement, nous avons besoin d'une fonction qui permet d'accéder à l'objet représenté par le concept, l'objet référé, afin de l'utiliser. Par exemple, nous aimerions accéder à l'objet dessin représenté par le concept [Dessin : JoliPaysage], ou encore être capable de manipuler le graphe représenté par le concept [Graph : [Cat]->(on)->[Mat] ].

Soit  $\omega$  une telle fonction.

**Définition 23** *La fonction  $\omega$  est définie de  $\mathcal{C} \rightarrow \mathcal{E}$  où  $\mathcal{C}$  est l'ensemble des concepts représentant les entités du système et  $\mathcal{E}$  est l'ensemble de tous les éléments référencés (internes et externes).*

Appliquée à un concept, elle retourne l'objet dont le concept est une abstraction.

$\omega([\text{Dessin} : \text{JoliPaysage}]) =$  
  
 $\omega([\text{Graph} : [\text{Cat}] \rightarrow (\text{on}) \rightarrow [\text{Mat}]] = [\text{Cat}] \rightarrow (\text{on}) \rightarrow [\text{Mat}]$

L'exemple avec l'opérateur  $\tau$  devient :

IF :  $[\text{Type} : *t_1] \rightarrow (\text{subt}) \rightarrow [\text{Type} : *t_2]$   
 THEN :  $[\omega([\text{Type} : *t_1]) : \forall *x] [\omega([\text{Type} : *t_2]) : ?x]$

Pour simplifier l'écriture  $\omega([\text{Type} : *t_1])$  pourra être noté tout simplement  $\omega t_1$ .

La règle s'écrit alors :

IF :  $[\text{Type} : *t_1] \rightarrow (\text{subt}) \rightarrow [\text{Type} : *t_2]$   
 THEN :  $[\omega t_1 : \forall *x] [\omega t_2 : ?x]$

De la même façon, la règle de traduction dans un diagramme E-R peut s'écrire :

IF :  $[\text{Type} : *t_1] \rightarrow (\text{arg1}) \rightarrow [\text{Relation} : *r] \leftarrow (\text{arg2}) \leftarrow [\text{Type} : *t_2]$   
 THEN :  $[\omega t_1] \rightarrow (\omega r) \rightarrow [\omega t_2]$

La fonction  $\omega$  est plus générale que les opérateurs  $\tau$  et  $\rho$  et peut être utilisée en lieu et place de ces opérateurs.

## Exemples

À titre d'exemple d'utilisation de la fonction  $\omega$  pour la métamodélisation, nous donnons ci-dessous la définition, sous forme de graphes conceptuels, de la transitivité, de la symétrie, de l'anti-symétrie et de la réflexivité d'une relation.

**Transitivité.** La figure 3.56 montre le graphe définissant une relation transitive. Une relation  $\mathcal{R}$  est transitive si et seulement si :

$$x\mathcal{R}y \wedge y\mathcal{R}z \Rightarrow x\mathcal{R}z \quad (3.1)$$

**Symétrie.** La figure 3.57 montre le graphe définissant une relation symétrique. Une relation  $\mathcal{R}$  est symétrique si et seulement si :

$$x\mathcal{R}y \Rightarrow y\mathcal{R}x \quad (3.2)$$



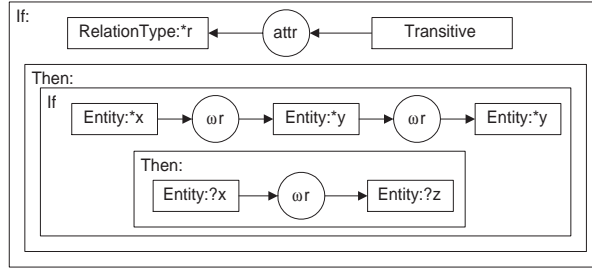


FIG. 3.56: Règle de définition d'une relation transitive.

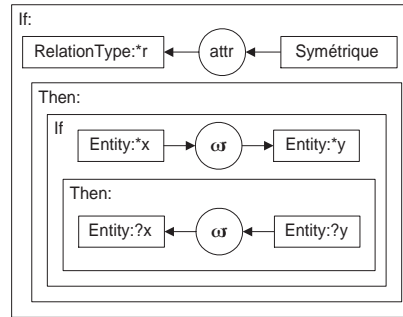


FIG. 3.57: Règle de définition d'une relation symétrique.

La figure 3.58 montre le graphe définissant une relation anti-symétrique. Une relation  $\mathcal{R}$  est anti-symétrique si et seulement si :

**Anti-symétrie.**

$$x\mathcal{R}y \wedge y\mathcal{R}x \Rightarrow x = y \quad (3.3)$$

ce qui est équivalent à

$$\neg(x\mathcal{R}y \wedge y\mathcal{R}x \wedge x \neq y) \quad (3.4)$$

**Réflexivité.** La figure 3.59 montre le graphe définissant une relation réflexive. Une relation  $\mathcal{R}$  définie sur T1 est réflexive si et seulement si :

$$\forall x \in T1, x\mathcal{R}x \quad (3.5)$$

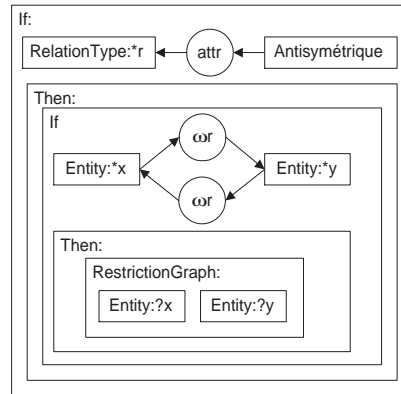


FIG. 3.58: Règle de définition d'une relation anti-symétrique.

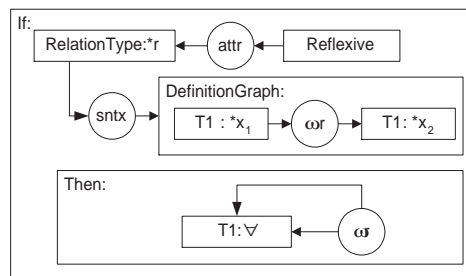


FIG. 3.59: Règle de définition d'une relation réflexive.

### 3.4 Contexte

Cette section présente en détail le rôle fondamental joué par les contextes. Les contextes sont indispensables pour segmenter l'information. Ils définissent des espaces de validité pour les assertions.

Les contextes ont été utilisés dans différents domaines par la communauté travaillant avec les graphes contextuels. Les contextes sont importants pour la définition de la connaissance [28, 29, 30], la structuration de la connaissance [59], le traitement de la langue naturelle [22, 55, 58, 57], la représentation des relations temporelles [55] et dans les systèmes multi-agents [56] et les systèmes objets [68].

Bien que les contextes aient été souvent utilisés, il n'y a pas de consensus sur leur utilisation. Dans le standard en cours d'élaboration [19], un contexte est défini comme un concept dont le référent est un graphe conceptuel. Nous avons étendu cette définition (3.2.4) en distinguant le graphe contextuel descriptif du graphe contextuel

qui détermine la validité du graphe conceptuel descriptif. Nous développons ci-dessous cette définition ainsi que les opérateurs pour la manipulation des contextes.

Un graphe  $g$  représentant une proposition vraie est appelé une assertion. Excepté pour les vérités universelles, la valeur de vérité d'une proposition dépend d'autres assertions. Ces assertions définissent les conditions dans laquelle la proposition est vraie. Ces assertions spécifient les situations où le graphe  $g$  est une assertion. L'ensemble de toutes les situations où le graphe  $g$  est une assertion est appelé sa *portée*. Une assertion ne peut être établie sans préciser sa portée. Les graphes sont donc toujours définis en relation à une situation. Par défaut, dans un système basé sur les graphes conceptuels, il existe une feuille d'assertion, dite *feuille d'assertion universelle*, toutes les vérités universelles sont faites sur la *feuille d'assertion universelle* qui représente la vérité universelle, c'est-à-dire, les graphes dont la valeur de vérité ne dépend pas d'autres conditions. Comme toutes les assertions sont établies en relation avec une situation (qui peut être la feuille d'assertion universelle), il est nécessaire de lier d'une manière formelle une proposition à l'ensemble des conditions dans lesquelles cette proposition est une assertion. Cet ensemble de conditions est appelé un *contexte*.

Un contexte est une feuille d'assertion dont l'existence dépend de graphes conceptuels qui décrivent les conditions de son existence. Tous les graphes du contexte dépendent des mêmes conditions. L'assertion "Marie pense que Pierre l'aime" est universellement vraie. Mais est-ce que Pierre aime réellement Marie ? Nous ne pouvons rien affirmer à ce sujet, excepté que "Pierre aime Marie" dans les pensées de Marie. Les pensées de Marie forment un contexte dans lequel la proposition "Pierre aime Marie" est vraie. La figure 3.60 montre l'assertion ; Les figures 3.61 et 3.62 montrent respectivement l'intension et l'extension du contexte associé. Les graphes d'intension et d'extension peuvent être dérivés automatiquement à partir de l'assertion de la figure 3.60.

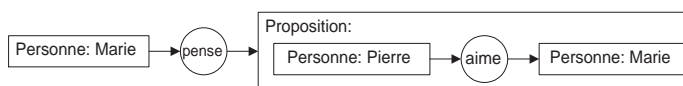


FIG. 3.60: La proposition "Marie pense que Pierre l'aime".

Un autre exemple présenté dans la figure 3.63 est la proposition suivante : "Marie pense que Pierre est beau".

Les contextes peuvent être emboîtés, comme dans la figure 3.64, qui présente l'as-



FIG. 3.61: L'intension du contexte des pensées de Marie.



FIG. 3.62: L'extension du contexte des pensées de Marie.

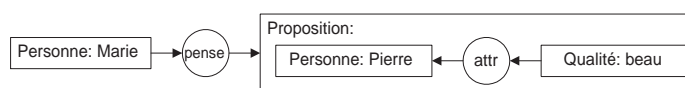


FIG. 3.63: La proposition "Marie pense que Pierre est beau".

sersion suivante : "Marie pense que Pierre pense qu'il est beau".

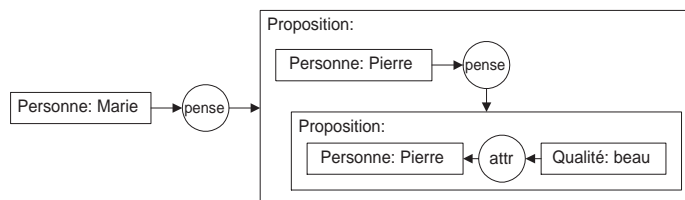


FIG. 3.64: La proposition "Marie pense que Pierre pense qu'il est beau".

Les graphes conceptuels des propositions des trois exemples (figures 3.60, 3.63 et 3.64) peuvent être regroupés dans une seule proposition comme illustrée par la figure 3.65.

La proposition représente la feuille d'assertion correspondant aux pensées de Marie. Le graphe conceptuel " Marie pense ..." est la condition d'existence de la feuille d'assertion. Le contexte des pensées de Marie est composé de deux parties : les conditions et les graphes vrais sous ces conditions.

Les contextes emboîtés peuvent être représentés d'une manière similaire. Le contexte de la figure 3.66 est la représentation de la figure 3.64, le contexte présente ce que Marie pense que Pierre pense.

Comme illustré dans les figures 3.65 et 3.66, un contexte est spécifié par deux parties : une *intension* qui décrit les conditions sous lesquelles les propositions sont vraies, et une *extension* qui est composée des graphes qui sont vrais sous ces condi-

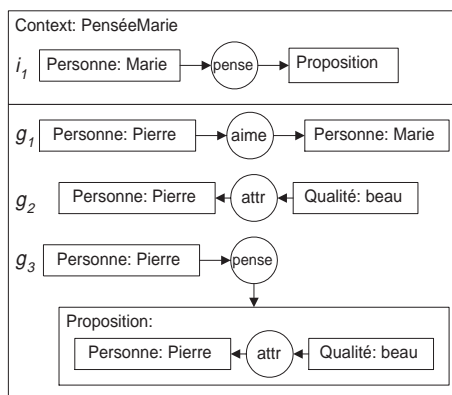


FIG. 3.65: Le contexte des pensées de Marie.

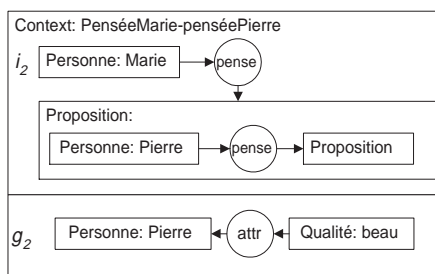


FIG. 3.66: Le contexte des pensées de Marie au sujet des pensées de Pierre.

tions. Un graphe peut apparaître dans l'extension de plus d'un contexte, comme par exemple le graphe représentant la proposition "Pierre est beau" qui apparaît dans le contexte des pensées de Marie et le contexte des pensées de Marie au sujet des pensées de Pierre. Si Pierre pense qu'il aime Marie, alors le contexte de la figure 3.67 montre les pensées de Pierre, et le graphe représentant la proposition "Pierre aime Marie" se trouve dans l'extension des pensées de Marie et dans l'extension des pensées de Pierre.

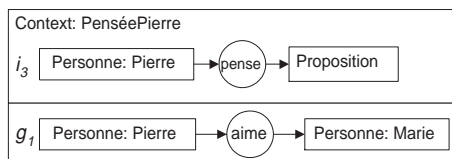


FIG. 3.67: Le contexte des pensées de Pierre.

D'une manière formelle (section 3.2.4) un contexte  $C$  est spécifié par deux ensembles de graphes  $T$  et  $G$ .  $T$  définit les conditions sous lesquelles  $C$  existe, et  $G$  est l'ensemble des graphes vrais dans ce contexte. Nous avons donc pour un contexte  $C$ ,  $C = \langle T, G \rangle$  où  $T$  et  $G$  sont respectivement l'intension et l'extension du contexte  $C$ . Soient  $intn$  la fonction qui associe un contexte à son intension et  $extn$  la fonction qui associe un contexte à son extension, nous pouvons écrire  $C = \langle T, G \rangle = \langle intn(C), extn(C) \rangle$ .

D'une part, nous pouvons remarquer qu'un graphe peut être vrai dans  $extn(C)$  sans avoir été défini explicitement dans le contexte  $C$ . En effet, si un graphe  $g$  est défini dans un autre contexte mais si le graphe  $g$  est une généralisation d'un graphe de  $extn(C)$ , alors  $g$  est vrai dans le contexte  $C$  et fait partie de  $extn(C)$ .  $extn(C)$  est défini comme la fermeture transitive par la relation de généralisation de tous les graphes définis dans  $C$ .

D'autre part, si deux contextes  $C_i$  et  $C_j$  sont tels que leurs graphes d'intension sont dans une relation de spécialisation  $intn(C_j) \leq intn(C_i)$ , alors les graphes vrais dans  $C_j$  sont vrais dans  $C_i$ . Nous avons donc l'inclusion suivante :  $extn(C_j) \subseteq extn(C_i)$ .  $extn(C_i)$  est donc plus que les graphes définis dans le contexte  $C_i$ ,  $extn(C_i)$  est l'union des graphes définis dans le contexte  $C_i$  et des graphes définis dans les contextes  $C_j$  tels que  $intn(C_j) \leq intn(C_i)$ . Un graphe  $g$  de  $extn(C_i)$  est une assertion de  $C_i$ , peu importe qu'il ait été défini dans  $C_i$  ou dans un contexte dont le graphe d'intension est dérivé du graphe d'intension de  $C_i$ .

Nous avons vu que la relation de spécialisation s'appliquait aussi bien aux graphes d'intension que d'extension pour calculer l'extension d'un contexte. La définition de portée définie pour un graphe comme l'ensemble des situations où le graphe est une assertion peut être étendue à un ensemble de graphes. La portée d'un ensemble de graphes  $G$  est l'ensemble des contextes dont l'extension inclut  $G$ . Soit  $P(G)$  la portée de  $G$  alors  $P(G) = \{C_i \mid G \subseteq extn(C_i)\}$ . La portée d'un graphe  $g$  est alors  $P(\{g\})$ .

La portée d'un ensemble de graphes est l'ensemble des contextes pour lesquels cet ensemble de graphes est un ensemble d'assertions. Considérons maintenant l'ensemble des graphes d'intension de la portée d'un ensemble de graphes. Soit  $I^*$  la fonction qui à un ensemble de graphes  $G$  associe cet ensemble de graphes d'intension. Nous avons l'équation suivante :

$$I^*(G) = \cup_i intn(C_i) \mid G \subseteq extn(C_i) \quad (3.6)$$

Inversement, nous pouvons définir l'ensemble des graphes  $G$  qui sont des assertions pour un ensemble de contextes dont  $T$  est l'ensemble des graphes d'intension. L'équation (3.7) ci-dessous introduit la fonction  $E^*$  qui associe à un ensemble de graphes d'intension  $T$  l'ensemble des graphes qui sont des assertions pour les contextes spécifiés par  $T$ .

$$E^*(T) = \cap_i \text{extn}(C_i) \mid \text{intn}(C_i) \subseteq T \quad (3.7)$$

Nous avons défini deux fonctions qui permettent d'interroger la mémoire d'entreprise constituée de graphes conceptuels. La fonction  $I^*$  nous permet de connaître l'intension d'un ensemble d'assertions et la fonction  $E^*$  qui nous permet inversement de connaître l'extension, c'est-à-dire, les assertions pour un ensemble de graphes d'intension.

Nous pouvons maintenant adapter les notions présentées dans [79] en définissant la notion de contexte formel. Un *contexte formel*  $C' = \langle T, G \rangle$  est un contexte dont l'intension  $T$  est exactement égale à l'ensemble  $I^*(G)$  des graphes d'intension de l'extension du contexte et dont l'extension  $G$  est exactement égale à l'ensemble  $E^*(T)$  des graphes qui sont des assertions pour l'intension du contexte. Nous avons donc l'équation suivante :

$$C^* = \langle T, G \rangle \text{ avec } T = I^*(G) = \text{Intn}(C^*) \text{ et } G = E^*(T) = \text{extn}(C^*) \quad (3.8)$$

Il existe un ordre partiel d'inclusion sur les graphes d'intension, et il existe un ordre partiel d'inclusion sur les extensions. À partir de ces relations d'ordre partiel, nous pouvons construire le treillis formé par l'ensemble des contextes formels. La figure 3.68 est le treillis construit à partir des exemples présentés dans les figures 3.65, 3.66 et 3.67.

Il est intéressant de noter que le treillis peut être calculé automatiquement. En effet, toutes les extensions  $\text{extn}(C_i)$  peuvent être calculées à partir des assertions définies dans les contextes  $C_i$  d'une manière automatique, et de la même façon tous les graphes d'intension  $\text{intn}(C_i)$  peuvent être calculés sans intervention à partir des graphes présents dans le système. La mémoire d'entreprise peut donc être automatiquement structurée sur une base sémantique telle qu'exprimée par les assertions qu'elle contient.

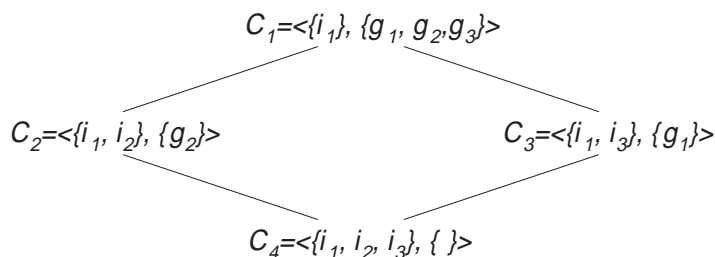


FIG. 3.68: Le treillis de contextes formels.

Nous verrons au chapitre suivant que la notion de contexte est un élément fondamental pour la recherche d'informations dans la base de connaissance. Les contextes et leurs structures sont des moyens de navigation dans les connaissances.

### 3.5 Sommaire

Nous venons de voir dans ce chapitre la partie langage de notre modèle comme illustré par la figure 3.69.

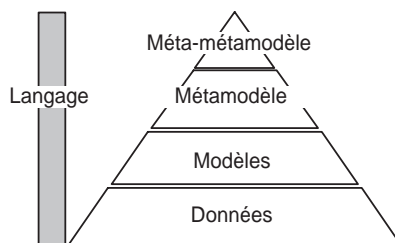


FIG. 3.69: La partie langage de la pyramide des modèles.

Nous avons présenté dans ce chapitre les principes qui sous-tendent notre langage :

1. Les graphes ne sont pas nécessairement connexes.
2. L'ensemble des types de concepts est un sous-ensemble du treillis engendré par la fermeture des types de concepts par les opérations de conjonction (intersection) et disjonction (union).
3. Les graphes sont sous forme normale. Deux concepts identiques représentant la même entité, sont systématiquement joints. Deux graphes ou parties de graphes ayant un concept commun sont fusionnées pour n'en faire qu'un seul ce qui



implique aussi une étape de simplification où toute relation dupliquée est systématiquement éliminée du nouveau graphe.

4. L'opération de projection est injective. Si deux concepts sont distincts alors ils se projettent sur des concepts distincts.
5. Tout concept doit évidemment vérifier le graphe de spécification de son type. De plus, pour qu'une relation conceptuelle puisse être établie entre deux concepts, il faut nécessairement que cette relation apparaisse dans au moins un des graphes de spécification des types des concepts concernés.
6. Seules les propositions réputées vraies sont insérées dans la base de connaissance. Toute proposition identifiée comme fausse est rejetée.

Nous avons vu les opérations fondamentales des graphes conceptuels qui permettent de dériver de nouveaux graphes à partir de graphes existants.

Nous avons présenté les éléments du langage en utilisant le langage lui-même. Le formalisme des graphes conceptuels permet de représenter dans un système de connaissances l'univers du discours qui est intérieur ou extérieur à ce système. Nous avons également défini une fonction de correspondance  $\omega$  qui permet d'accéder à partir d'un concept à l'élément représenté. Cette fonction n'a évidemment de sens que pour les éléments qui sont à l'intérieur du système, comme par exemple accéder au graphe représenté par un concept de type `graph` ou encore accéder à l'entier 3 représenté par le concept `[Integer : 3]`. Enfin nous avons détaillé aussi le mécanisme de contexte qui permet de structurer la connaissance en fonction de domaine de validité.

Le prochain chapitre introduit les deux niveaux supérieurs de la hiérarchie, le méta-métamodèle et les métamodèles du modèle uniforme.

# Chapitre 4

## Méta-métamodèle et métamodèles

Ce chapitre présente comment les concepts définissant les deux niveaux supérieurs de la pyramide des modèles sont structurés. Le méta-métamodèle et les métamodèles sont basés sur le langage présenté dans le chapitre précédent. Ils sont une spécialisation de ce langage. Les éléments du méta-métamodèle sont les concepts permettant de définir un métamodèle. Les éléments du métamodèle sont les concepts permettant de définir les modèles.

La figure 4.1 illustre les liens d'instance et de spécialisation entre les différentes sortes de types définis dans le modèle uniforme. D'une part, nous avons défini dans le langage la notion de type de concept qui permet de catégoriser des instances en spécifiant les relations conceptuelles que peuvent entretenir les instances. D'autre part, l'architecture en couches à quatre niveaux définit trois niveaux où peuvent apparaître des types : modèle, métamodèle et méta-métamodèle.

Les quatre éléments Concept Type, Data Type, Model Type, Meta Type peuvent être perçus comme types ou comme instances.

D'une part, en tant que types, il existe des liens de spécialisation entre Concept Type et les trois éléments Data Type, Model Type, et Meta Type. En effet, ces trois types sont des sous-types de Concept Type. Ce sont des types spécialisés qui apparaissent à un niveau précis de la pyramide. Ils héritent des propriétés de Concept Type ; leurs instances sont définies par des graphes de spécification.

D'autre part, il existe des liens d'instance entre Concept Type et les trois éléments Data Type, Model Type, et Meta Type. En effet, ce sont des types car ils ont des instances et ils sont spécifiés par des graphes de définition. La figure 4.1 présente une relation d'instance de Concept Type sur lui-même. En effet, Concept Type en tant que

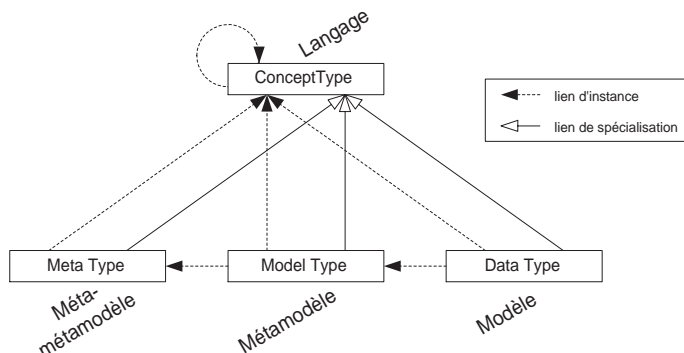


FIG. 4.1: Spécialisation et instanciation des types.

type est instance de lui-même. Ce type de lien peut amener à des paradoxes comme "l'ensemble de tous les ensembles". Cependant, Concept Type est le seul élément du langage à avoir cette propriété et un système implémentant le modèle uniforme devra en tenir compte.

Nous présentons maintenant dans ce chapitre les éléments du méta-métamodèle et des métamodèles. Le langage présenté dans le chapitre précédent est utilisé à tous les niveaux de la pyramide et fait donc partie conceptuellement du méta-métamodèle, des métamodèles, des modèles et des données comme illustré par la figure 4.2. Cependant nous ne présenterons pas dans chacune des sections suivantes les éléments du langage au complet. Nous ne présenterons que les éléments qui sont spécialisés pour chacun des niveaux.

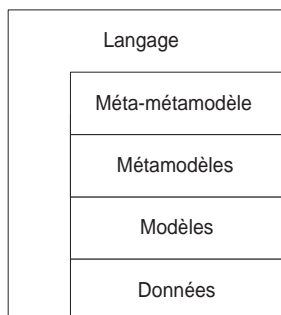


FIG. 4.2: Le langage est utilisé à tous les niveaux.

## 4.1 Méta-métamodèle

Cette section présente le méta-métamodèle de notre modèle uniforme. Le méta-métamodèle est la couche supérieure dans une architecture à quatre couches. Elle permet de modéliser les formalismes de représentation des connaissances.

### 4.1.1 Introduction

Le méta-métamodèle décrit les entités rencontrées dans un métamodèle. Un métamodèle étant par définition un modèle (de niveau supérieur), on retrouve les éléments permettant la catégorisation des entités du niveau inférieur. Les deux principaux éléments sont donc `MetaConType` pour modéliser les types de concept du métamodèle et `MetaRelType` pour modéliser les types de relation du métamodèle.

Les figures 4.3, 4.4 et 4.5 présentent les éléments du méta-métamodèle et les relations qu'ils entretiennent. La figure 4.3 illustre la hiérarchie des éléments du méta-métamodèle. Les éléments du méta-métamodèle sont dérivés des éléments de base du langage des graphes conceptuels. Nous retrouvons donc les notions de concept, relation, graphe conceptuel, type de concept et type de relation.

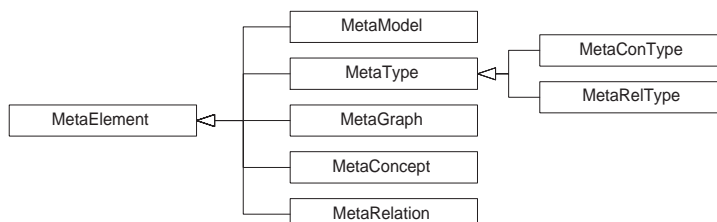


FIG. 4.3: Hiérarchie des types du méta-métamodèle.

Les autres éléments du langage, comme les graphes de définition de types, sont également éléments du méta-métamodèle mais ne sont pas répétés ici car leurs définitions ne sont pas spécialisées. En effet, ils sont indépendants du niveau où ils sont utilisés. Nous ne présentons ici que les types dépendants du niveau méta-métamodèle.

La figure 4.4 montre les liens de spécialisation des éléments du méta-métamodèle avec les éléments du langage. Les méta-graphes, les méta-concepts et les méta-relations sont respectivement des graphes, des concepts et des relations qui apparaissent dans un graphe de métamodèle. De même les méta-types de concept et les méta-types

de relation sont respectivement des types de concept et des types de relation qui sont définis dans un métamodèle.

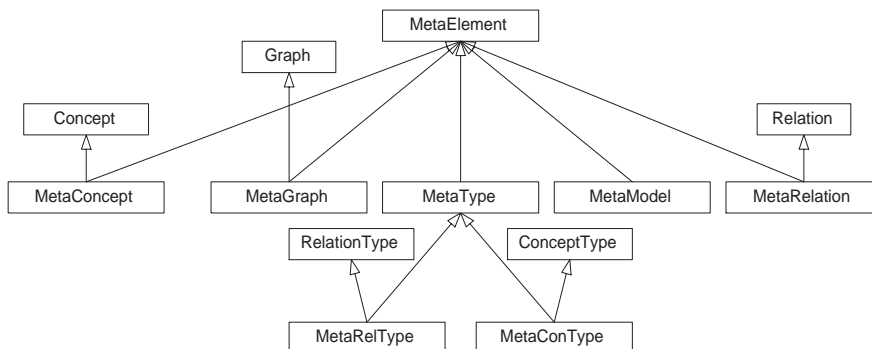


FIG. 4.4: Les liens entre langage et méta-métamodèle.

La figure 4.5 montre les liens qui organisent les types du méta-métamodèle. Comme dans le langage, un méta-graphe est composé de méta-concept et de méta-relation ; les méta-concepts sont instances de méta-type de concepts et les méta-relations sont instances de méta-types de relation.

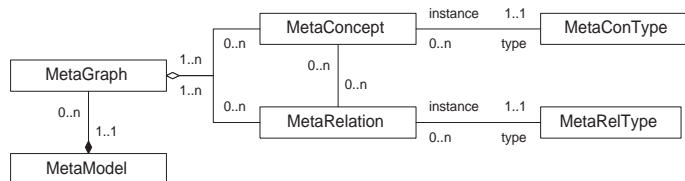


FIG. 4.5: Le méta-métamodèle (formalisme UML).

#### 4.1.2 Éléments du méta-métamodèle

Nous détaillons ci-dessous chacun des types du méta-métamodèle présentées dans la figure 4.3.

##### Méta-élément (MetaElement)

Les méta-éléments sont les constituants du métamodèle.

### Méta-graphe (MetaGraph)

Les méta-graphes sont les graphes apparaissant au niveau métamodèle.

**Définition 24** *Un méta-graphe est un graphe exprimant des connaissances sur les éléments du métamodèle.*

Un méta-graphe est un graphe dont les éléments peuvent être des méta-concepts et des méta-relations.

### Métamodèle (MetaModel)

Les métamodèles sont les formalismes de la représentation des connaissances.

**Définition 25** *Un métamodèle est un ensemble de graphes décrivant les éléments du métamodèle.*

Le nom de métamodèle peut porter à confusion. Contrairement à ce que le nom pourrait laisser croire un métamodèle n'a pas pour instance des modèles. Un métamodèle est un ensemble de méta-éléments décrivant les éléments des modèles. Un métamodèle est un ensemble de graphes exprimant des connaissances sur les éléments du métamodèle. Par exemple, l'ensemble des graphes décrivant les composantes de UML est le métamodèle de UML.

### Méta-concept (MetaConcept)

Les méta-concepts sont les concepts apparaissant dans les graphes du niveau métamodèle.

**Définition 26** *Un méta-concept est un concept du niveau métamodèle.*

### Méta-relation (MetaRelation)

Les méta-relations sont les relations apparaissant dans les graphes du niveau métamodèle.

**Définition 27** *Une méta-relation est une relation du niveau métamodèle.*

### Méta-type (MetaType)

Les méta-types sont les types qui apparaissent au niveau métamodèle.

**Définition 28** *Un méta-type est un type du niveau métamodèle.*

### Méta-type de concept (MetaConType)

Les méta-types de concept (MetaConType) sont les types de concept apparaissant dans les graphes du niveau métamodèle. Les instances de ces méta-types de concept sont des concepts du niveau modèle.

**Définition 29** *Un méta-type de concept est un type de concept du niveau métamodèle. Il définit un ensemble de concepts qui sont utilisés au niveau modèle.*

### Méta-type de relation (MetaRelType)

Les méta-types de relation (MetaRelType) sont les types de relation apparaissant dans les graphes du niveau métamodèle. Les instances de ces métatypes de relation sont des relations du niveau modèle.

**Définition 30** *Un méta-type de relation est un type de relation du niveau métamodèle. Il définit un ensemble de relations conceptuelles qui sont utilisées au niveau modèle.*

## 4.2 Métamodèles

Cette section présente les métamodèles de notre modèle uniforme.

### 4.2.1 Introduction

Les métamodèles sont les formalismes pour la représentation des connaissances de l'entreprise, en particulier pour la représentation des ontologies du domaine. Les figures 4.6 et 4.7 présentent les éléments racines communs aux différents métamodèles et les éléments différenciateurs des différents métamodèles.

La figure 4.6 illustre la hiérarchie des types du métamodèle.

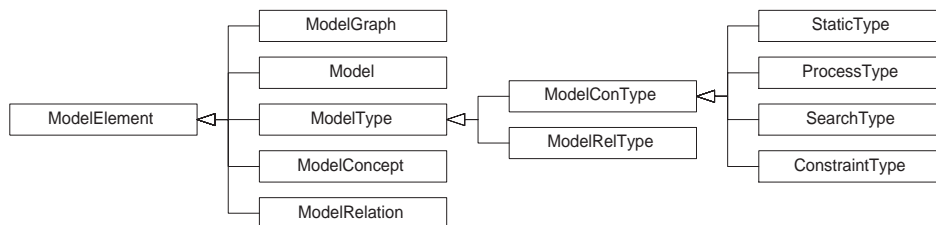


FIG. 4.6: Hiérarchie des types du méta-métamodèle.

La figure 4.7 montre les liens qui structurent les types du métamodèle.

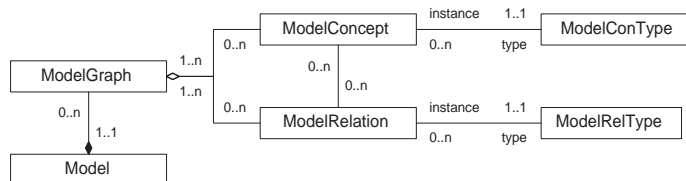


FIG. 4.7: Le métamodèle (formalisme UML).

#### 4.2.2 Noyau commun

Cette section introduit les éléments racines et communs du métamodèle.

##### Élément de modèle (ModelElement)

Les éléments de modèle sont les constituants des modèles.

##### Modèle (Model)

Les modèles représentent d'une manière générique et formalisée les connaissances statiques et dynamiques de l'entreprise.

**Définition 31** *Un modèle est un ensemble de graphes décrivant les composantes du modèle.*

Un modèle est un graphe dont les éléments sont des graphes définissant les éléments du modèle.

##### Graphe de modèle (ModelGraph)

Les graphes de modèle sont les graphes apparaissant au niveau modèle.

**Définition 32** *Un graphe de modèle est un graphe du niveau modèle.*

Un graphe de modèle est un graphe dont les éléments sont des concepts de modèle et des relations de modèle.



### Concept de modèle (ModelConcept)

Les concepts de modèle sont les concepts apparaissant dans les graphes du niveau modèle.

**Définition 33** *Un concept de modèle est un concept du niveau modèle.*

### Relation de modèle (ModelRelation)

Les relations de modèle sont les relations apparaissant dans les graphes du niveau modèle.

**Définition 34** *Une relation de modèle est une relation du niveau modèle.*

### Type de concept de modèle (ModelConType)

Les types de concept de modèle (ModelConType) sont les types de concept apparaissant dans les graphes du niveau modèle. Les instances de ces types de concept sont des concepts du niveau données.

**Définition 35** *Un type de concept de modèle est un type de concept du niveau modèle.*

### Type de relation de modèle (ModelRelType)

Les types de relation de modèle (ModelRelType) sont les types de relation apparaissant dans les graphes du niveau modèle. Les instances de ces types de relation sont des relations du niveau données.

**Définition 36** *Un type de relation de modèle est un type de relation du niveau modèle.*

#### 4.2.3 Métamodèle de structure

Cette section introduit les éléments permettant de structurer les types de connaissances statiques de l'entreprise. Nous distinguons trois catégories de types de connaissances :

- les types de connaissances qui concernent la structure de l'entreprise,
- les types de connaissances qui concernent l'information de l'entreprise.
- les types de connaissances qui concernent les objets d'affaires de l'entreprise.

Nous détaillons ci-dessous les types concernés par ces catégories.

### Type de structure (**StructureType**)

Les éléments de cette catégorie concernent les types de concept structurant les types d'entités de l'entreprise ou impliquant l'entreprise.

**Définition 37** *Un type de structure est un type de concept modélisant la partie structurelle d'une entreprise.*

Des exemples de types de structure sont :

- les organigrammes qui structurent les ressources humaines,
- les structures juridiques qui organisent les entités juridiques,
- les structures organisationnelles qui définissent les notions de siège social, succursale, département, service, ...
- les structures de l'environnement de l'entreprise (concurrents, gouvernement,...).

### Type d'information (**InformationType**)

Les types définis dans cette catégorie permettent de modéliser les objets de type informationnel, c'est à dire, les objets non tangibles.

**Définition 38** *Un type d'information est un type de concept modélisant la partie informationnelle d'une entreprise.*

Des exemples de types d'information sont :

- direction, mission, but, objectif,
- marché,
- frontière,
- hypothèse,
- techniques, savoir-faire.

### Type d'objets d'affaires (**BusinessObjectType**)

Les types d'objets d'affaires permettent de modéliser les objets tangibles. Les objets tangibles sont les objets physiques ou les objets produisant des objets tangibles.

**Définition 39** *Un type d'objets d'affaires est un type de concept modélisant la partie tangible d'une entreprise.*

Des exemples de types d'objets d'affaires sont :

- produits manufacturés,
- services,
- documents,
- logiciels, fichiers électroniques.

#### 4.2.4 Métamodèle de dynamique

Cette section introduit les éléments permettant de structurer la connaissance dynamique de l'entreprise. La terminologie a été fortement influencée par l'approche systémique présenté dans [31].

##### **Type de système (SystemType)**

Les types de système permettent de modéliser les différents systèmes qui composent l'univers de l'entreprise. L'univers de l'entreprise peut être vu comme un ensemble de systèmes interagissant entre eux.

**Définition 40** *Un type de système est un type de concept modélisant la partie systémique d'une entreprise ainsi que son environnement.*

Des exemples de types de systèmes sont :

- entreprise,
- environnement,
- gouvernement.

##### **Type de participant de système (SystemParticipantType)**

Les types de participant de système permettent de modéliser les objets qui sont associés d'une manière interne ou externe aux systèmes.

**Définition 41** *Un type de participant de système est un type de concept modélisant les entités impliquées dans un système.*

Des exemples de types de participant de système sont :

- concurrent,
- département,
- matières premières.

### Type de modèle de processus (ProcessModelType)

Les types de modèles de processus permettent de modéliser les processus complexes. Un processus complexe est décrit par un modèle qui montre les activités qui doivent être exécutées.

**Définition 42** *Un type de modèle de processus est un type de concept modélisant la séquence d'activités permettant de réaliser un processus complexe.*

Des exemples de types de modèle de processus sont :

- comment octroyer un prêt hypothécaire,
- comment rédiger un appel d'offres,
- comment fabriquer un téléviseur.

### Type d'événement (EventType)

Les types d'événement permettent de modéliser la fin des activités. Un événement marque la fin d'une activité. Les instances d'événement permettent de conserver l'historique de la réalisation d'un processus.

**Définition 43** *Un type d'événement est un type de concept modélisant la fin d'une activité.*

Des exemples de types d'événement sont :

- formulaire rempli,
- contrat signé,
- produit livré.

### Type de processus (ProcessType)

Les types de processus permettent de modéliser les transformations se réalisant à l'intérieur d'un système.

**Définition 44** *Un type de processus est un type de concept modélisant les activités.*

Des exemples de types de processus sont :

- remplir un formulaire,
- signer un contrat,
- octroyer un prêt,
- interviewer un candidat.

### Type de participant (ParticipantType)

Les types de participant modélisent les types d'entités (ressources matérielles, ressources humaines, produits) impliquées dans la réalisation des processus.

**Définition 45** *Un type de participant est un type de concept modélisant les entités impliquées dans un processus.*

Des exemples de types de participant sont :

- architecte,
- commis,
- comptable.

### Type d'état (StateType)

Les types d'état modélisent les types d'étapes caractérisant la vie d'une entité. Les états sont représentés par des conditions particulières portant sur les propriétés ou sur les relations des entités.

**Définition 46** *Un type d'état est un type de concept modélisant les étapes ou des conditions particulières dans la vie d'un objet d'affaires.*

Des exemples de types d'état sont :

- payée (pour une facture),
- vérifié (pour un état de compte),
- complété (pour un formulaire).

### Type de condition (ConditionType)

Les types de condition modélisent les types de proposition qui décrivent des situations ou des circonstances qui entraînent la réalisation d'une activité ou qui sont le résultat de cette activités.

**Définition 47** *Un type de condition est un type de concept modélisant les circonstances auxquelles est subordonné l'accomplissement d'une action, ou soumise la production d'un phénomène. (adapté de Le Petit Larousse illustré 1999)*

Des exemples de types de condition sont :

- pré condition,
- post condition.

### 4.2.5 Métamodèle de recherche

Cette section introduit les éléments permettant de rechercher dans les connaissances. La recherche de l'information se fait par l'expression de critères de sélection. Ce domaine a été largement étudié par la communauté des bases de données qui a défini des langages de manipulation de données tel que SQL. Notre objectif est de définir un langage d'interrogation qui répond aux besoins d'une mémoire d'entreprise et qui utilise le formalisme des graphes conceptuels. Des travaux ont déjà été menés dans ce domaine [70, 50], et nous les étendons. Le langage que nous définissons ici doit permettre d'interroger les connaissances quel que soit le niveau de modélisation où ces connaissances se trouvent et ceci d'une manière uniforme.

#### Introduction

Comme le rappelle Gardarin dans [33] les chercheurs ont d'abord essayé de modéliser les langages d'interrogation de bases de données par la logique. Ainsi sont nés les langages d'interrogation de bases de données comme QBE (Query-By-Example) [81] basé sur le calcul relationnel de domaines ou comme SQL basé sur le calcul relationnel de tuples [17]. Comme ces langages d'interrogation de bases de données, le formalisme des graphes conceptuels est basé sur la logique du premier ordre.

Une formule logique prend ses valeurs de vérité sur un domaine de discours défini par [33] comme l'ensemble des objets sur laquelle la formule prend valeur par interprétation des constantes comme des objets particuliers, des variables comme des objets quelconques, des prédicats comme des relations entre objets et des fonctions comme des relations particulières entre objets.

Un graphe conceptuel est équivalent à une formule logique. La phrase "Un chat est sur un tapis" s'exprime par le graphe conceptuel

$$[\text{Chat} : *x] \rightarrow (\text{sur}) \rightarrow [\text{Tapis} : *y] \quad (4.1)$$

qui est équivalent à la formule logique

$$\exists x, \exists y, \text{Chat}(x) \wedge \text{Tapis}(y) \wedge \text{sur}(x, y) \quad (4.2)$$

Répondre à la question "Quels sont les chats qui sont sur un tapis?" consiste à trouver les objets qui rendent la formule logique (4.2) vraie. Sur une base de données constituée de graphes conceptuels, cela consiste à trouver les graphes dérivés du graphe (4.1).

Interroger une base de données constituée de graphes conceptuels s'inscrit dans la problématique de l'interrogation de données semi-structurées. Des données semi-structurées sont des données dont le schéma est flexible, c'est à dire, que le modèle de données peut évoluer. Un exemple de données semi-structurées est un document XML. XML [74] (eXtensible Markup Language) est un standard pour l'échange d'information sur le Web proposé par le Consortium W3C. Un document XML peut être représenté par un graphe orienté. La figure 4.8 montre comment un document XML peut être représenté par un graphe conceptuel. La structure hiérarchique du document XML est représentée par un graphe orienté.

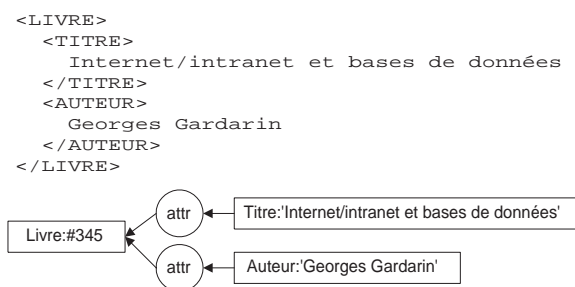


FIG. 4.8: Exemple de document XML et graphes conceptuels.

Des langages d'interrogation de documents structurés ont été proposés [3, 21, 10, 15, 45]. Parmi eux XML-QL [21] est dérivé de XML. Il utilise la syntaxe de XML pour interroger les documents. Basé sur le concept de canevas, XMLQL utilise un document XML dont certaines données sont remplacées par des variables que l'on cherche à instancier. La figure 4.9 montre un exemple de requête exprimée en XMLQL.

```

<LIVRE>
  <TITRE>
    $t
  </TITRE>
  <AUTEUR>
    Georges Gardarin
  </AUTEUR>
</LIVRE>

```

FIG. 4.9: 'Quels sont les livres écrit par Georges Gardarin?'.

Une particularité de ces langages d'interrogation, qui sont dans ce sens plus puissants que SQL, est la possibilité d'interroger les méta-données aussi bien que les données. La figure 4.9 présente la requête XMLQL permettant de retrouver les livres

où le nom de Georges Gardarin apparaît, qu'il soit auteur, rédacteur ou éditeur. La variable  $\$x$  remplace n'importe quelle balise XML.

```
<LIVRE>
  <TITRE>
    $t
  </TITRE>
  <$x>
    Georges Gardarin
  </$x>
</LIVRE>
```

FIG. 4.10: 'Quels sont les livres où apparaît Georges Gardarin?'

## Recherche dans les graphes conceptuels

Pour interroger les bases de graphes conceptuels, nous proposons un langage ayant les mêmes caractéristiques. L'interrogation se fait au travers de requêtes. Une *requête* est un graphe dont on essaie d'instancier les variables. Soit la question "Quels sont les chats qui sont sur un tapis?". Le graphe conceptuel suivant va nous aider à répondre à la question.

$$[\text{Chat} : *x] \rightarrow (\text{sur}) \rightarrow [\text{Tapis} : *y] \quad (4.3)$$

Afin d'évaluer les requêtes, nous définissons une fonction  $\sigma$  qui possède deux arguments : le premier est un ensemble de graphes et le deuxième est la requête à évaluer sur cet ensemble de graphes. La fonction  $\sigma$  est donc définie par l'équation 4.4 ci-dessous. La fonction retourne l'ensemble des projections de  $q$  dans les graphes de  $G$  selon une projection injective telle que définie à la section 3.2 (page 58).

$$\sigma(G, q) = \{g \in G \mid g = \pi(q)\} \quad (4.4)$$

Donnons un exemple d'utilisation de la fonction  $\sigma$ . Soit  $G$  l'ensemble de graphes présenté dans la figure 4.11. Et soit  $q_1$  la question "Quels sont les chats qui sont sur un tapis?" (voir équation 4.3).

Alors nous avons l'ensemble résultat  $G' = \sigma(G, q_1)$ . Il est présenté à la figure 4.12.

La question posée était "Quels sont les chats qui sont sur un tapis?". Le résultat  $G'$  n'est donc pas exactement le résultat attendu. Il faut donc se restreindre aux



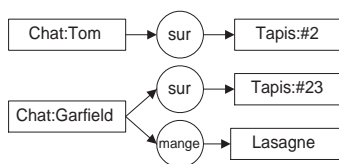


FIG. 4.11: Un ensemble de graphes à interroger.

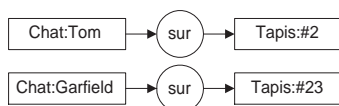
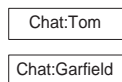
FIG. 4.12: Les graphes résultats  $G' = \sigma(G, q_1)$ .

FIG. 4.13: Les chats assis sur un tapis.

concepts qui identifient les chats. Soit  $q_2$  le graphe  $\boxed{\text{Chat:*x}}$ , alors le résultat demandé est  $G'' = \sigma(G', q_2)$  dont la figure 4.13 montre les éléments.

Nous avons donc deux requêtes emboîtées pour répondre à la question. L'ensemble des chats assis sur un tapis est obtenu par la double requête illustrée par l'équation

$$G'' = \sigma(\sigma(G, q_1), q_2) \quad (4.5)$$

## Recherche et contexte

Nous avons vu à la section 3.2 la notion de contexte qui permet de structurer la connaissance dans une mémoire d'entreprise ou plus généralement dans une base de connaissance. Nous allons voir ici comment interroger une base structurée [52].

L'évaluation d'une requête dépend du contexte dans lequel se fait l'évaluation. Nous devons donc préciser le contexte pour l'évaluation des requêtes. Le contexte est, soit le contexte universel, c'est à dire, la feuille d'assertion universelle, soit un contexte spécifique défini dans la base de connaissance. Nous avons vu qu'un contexte est composé de deux ensembles de graphes : l'extension qui est l'ensemble des assertions du contexte et l'intension qui est l'ensemble des conditions pour lesquelles les

graphes de l'extension sont vrais. Les requêtes exprimées sur les contextes peuvent porter sur l'intension ou sur l'extension. Deux fonctions sont donc nécessaires pour l'interrogation des contextes. Ces deux fonctions sont présentées dans les équations suivantes.

$$\delta_E(C^*, q) = \{g \in E(C^*) \mid g \leq q\} \quad (4.6)$$

$$\delta_I(C^*, q) = \{g \in I(C^*) \mid g \leq q\} \quad (4.7)$$

La fonction  $\delta_E$  recherche dans les graphes d'extension du contexte, tandis que la fonction  $\delta_I$  recherche dans les graphes d'intension du contexte. Le résultat de la fonction  $\delta_E$  est un ensemble de graphes d'assertion, et le résultat de la fonction  $\delta_I$  est un ensemble de graphes d'intension. Mais si nous voulons pouvoir naviguer dans le treillis formé par les contextes, nous devons reconstruire un contexte à partir d'un ensemble de graphes d'extension ou d'un ensemble de graphes d'intension. Pour cela nous introduisons deux nouvelles fonctions  $C_E$  (4.8) et  $C_I$  (4.9) qui reconstruisent un contexte respectivement à partir d'un ensemble de graphes d'extension et d'un ensemble de graphes d'intension. Ces fonctions utilisent les fonctions  $E^*$  et  $I^*$  définies au chapitre 3.2. Rappelons que  $E^*$  associe à un ensemble de graphes d'intension  $G$ , les graphes d'extension correspondants, et que  $I^*$  associe à un ensemble de graphes d'extension  $T$ , les graphes d'intension correspondants.

$$C_E(G) = \langle I^*(G), E^*(I^*(G)) \rangle \quad (4.8)$$

$$C_I(T) = \langle I^*(E^*(T)), E^*(T) \rangle \quad (4.9)$$

Nous allons illustrer la navigation dans la structure des contextes formels sur l'exemple donné dans le chapitre 3.2. La structure des contextes formels est rappelée dans la figure 4.14

Nous allons détailler la navigation présentée dans l'équation 4.10 qui correspond à la question "Est-ce que Pierre pense comme Marie, à savoir qu'il l'aime? ".

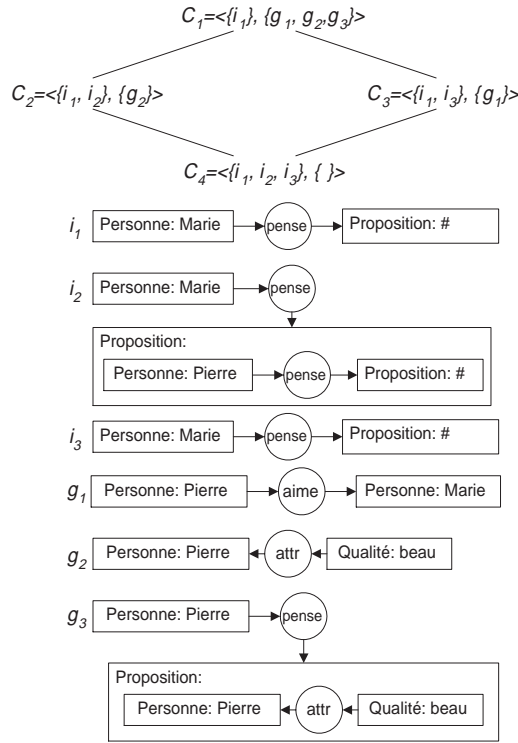


FIG. 4.14: La structure des contextes formels.

$$\delta_I(C_E(\delta_E(C_E(\delta_E(U, q_1)), q_2)), q_3) \quad (4.10)$$

Il y a trois questions imbriquées. La première question identifie les pensées de Marie, la deuxième question identifie le fait que Marie pense que Pierre l'aime, et enfin la dernière question identifie si Pierre a la même pensée.

La première question  $q_1 : [\text{Personne} : \text{Marie}] \rightarrow (\text{pense}) \rightarrow [\text{Proposition} : *p]$  est exécutée dans le contexte universel  $U$ . Le résultat est l'ensemble des trois graphes  $g_1, g_2, g_3$  qui constituent les pensées de Marie. Le résultat est donc  $S_1 = \delta_E(U, q_1) = \{g_1, g_2, g_3\}$ . Le contexte reconstruit à partir de  $S_1$  est le contexte  $C_E(S_1) = C_1$ .

La deuxième question  $q_2 : [\text{Personne} : \text{Pierre}] \rightarrow (\text{aime}) \rightarrow [\text{Personne} : \text{Marie}]$  est appliquée sur l'extension du contexte  $C_1$  et recherche si le graphe  $q_2$  se trouve dans les pensées de Marie. Le résultat est  $S_2 = \delta_E(C_1, q_2) = \{g_1\}$  et le contexte reconstruit à partir de  $S_2$  est le contexte  $C_E(S_2) = C_3$ .

Et la troisième question  $q_3 : [\text{Personne} : \text{Pierre}] \rightarrow (\text{pense}) \rightarrow [\text{Proposition} : *p]$

est appliquée sur l'intension du contexte  $C_2$ . Nous recherchons si le graphe  $q_3$  se trouve dans l'intension du contexte. Le résultat est  $S_3 = \delta_I(C_3, q_3) = \{i_3\}$ . La réponse à la question est donc oui, Pierre pense comme Marie qu'il l'aime.

La structure de contextes formels et le mécanisme de navigation dans la structure permettent de rapidement accéder aux informations pertinentes sans être obligé de parcourir toute la base de connaissance pour essayer de trouver les graphes qui conviennent.

Nous terminerons cette section en montrant que les deux fonctions  $\delta_E$  et  $\delta_I$  qui permettent de naviguer dans la structure des contextes formels peuvent s'exprimer en fonction de l'opération de restriction  $\sigma$  vue dans la section précédente. Basées sur la définition d'un contexte comme spécifié au chapitre 3.2, soient

$$q_E(C) : [\text{Context} : C] \leftarrow (\text{extn}) \leftarrow [\text{Graph} : *p]$$

la requête qui recherche les graphes d'extension d'un contexte  $C$  et

$$q_I(C) : [\text{Context} : C] \leftarrow (\text{intn}) \leftarrow [\text{Graph} : *p]$$

la requête qui recherche les graphes d'intension d'un contexte  $C$ . Alors nous pouvons écrire les deux équations suivantes qui donnent la définition de  $\delta_E$  et  $\delta_I$  en fonction de  $\sigma$ .

$$\delta_E(C, q) = \sigma(\sigma(U, q_E(C)), q) \quad (4.11)$$

$$\delta_I(C, q) = \sigma(\sigma(U, q_I(C)), q) \quad (4.12)$$

### Types de concept de recherche

Nous venons de voir la fonction permettant de rechercher dans les graphes conceptuels. Pour l'intégration de ce mécanisme de recherche dans le formalisme, il nous reste à définir les méta-concepts qui vont nous permettre d'exprimer des requêtes dans la mémoire d'entreprise. Nous présentons ci-dessous les trois types de concept nécessaires à cette fin.

### Graphe de recherche (QueryGraph)

Un graphe de recherche est une requête exprimée sur la base de connaissances.

**Définition 48** *Un graphe de recherche est un graphe dont on cherche les projections dans un ensemble de graphes de la base de connaissance.*

Un graphe de recherche est évalué sur un ensemble de graphes, et les résultats sont les graphes appartenant à cet ensemble de graphes qui sont des projections du graphe de requête.

### Espace de recherche (QuerySpace)

Un espace de recherche est un sous-ensemble des graphes de la base de connaissances. Ce sous-ensemble définit l'espace dans lequel le graphe de recherche va être évalué.

**Définition 49** *Un espace de recherche est un ensemble de graphes sur lesquels est évalué le graphe de recherche.*

### Espace de solution (SolutionSpace)

Un espace de solution est associé à un graphe de recherche et un espace de recherche. L'espace des solutions est constitué des graphes qui répondent à la requête. Plus précisément nous avons la définition suivante :

**Définition 50** *Un espace de solution est l'ensemble des graphes appartenant à un espace de recherche et qui sont des projections du graphe de recherche.*

### Fonction $\sigma$

La fonction  $\sigma$  permet d'associer à un espace de recherche et à un graphe de recherche l'espace de solution correspondant. La figure 4.15 présente les graphes de spécification de la fonction  $\sigma$ .

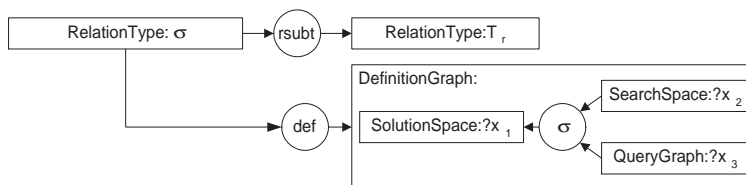


FIG. 4.15: Graphe de spécification de la fonction  $\sigma$ .

Nous venons de voir le métamodèle de recherche. Il définit formellement les notions et opérateurs liés à l'interrogation de la base de connaissance.

#### 4.2.6 Métamodèle de contrainte

Cette section introduit les éléments permettant de spécifier les contraintes sur la connaissance de l'entreprise et plus généralement sur la base de connaissance.

Basés sur les principes de notre modèle, il existe deux mécanismes de contraintes :

- Tout concept doit être conforme à la spécification de son type tout au long de son cycle de vie.
- Seules les relations définies dans les définitions de types de concept impliqués peuvent être établies.

Nous avons vu qu'un type de concept est spécifié par une définition qui détermine les relations que le concept doit avoir, des graphes de restriction qui déterminent les relations que le concept ne peut pas entretenir et enfin les graphes de règles qui permettent d'exprimer des contraintes plus sophistiquées que obligatoire (graphe de définition) ou interdit (graphe de restriction). L'utilisation des graphes de restriction et des graphes de règles permettent d'exprimer un grand nombre de contraintes.

Nous avons déjà dit qu'une mémoire d'entreprise ne traitait pas des objets opérationnels mais nous devons pouvoir y exprimer des règles d'affaires. Nous verrons au chapitre 5 que les deux mécanismes rappelés ci-dessus permettent d'exprimer toutes les contraintes que l'on peut trouver dans une mémoire d'entreprise.

### 4.3 Un exemple

Afin de bien comprendre l'intérêt de notre modèle uniforme nous terminons ce chapitre par un exemple de modélisation et métamodélisation de connaissances corporatives. Notre exemple porte sur une coopérative de vente au détail dont les clients doivent être membres pour passer des commandes. Nous présentons le cas d'une activité bien spécifique des tâches administratives de la coopérative : le traitement des demandes d'adhésion. Pour adhérer un membre doit remplir un formulaire. Le formulaire est traité par un gestionnaire qui émet une carte de membre pour le nouveau client. La figure 4.16 représente une instance de cette activité de traitement d'une demande d'adhésion.

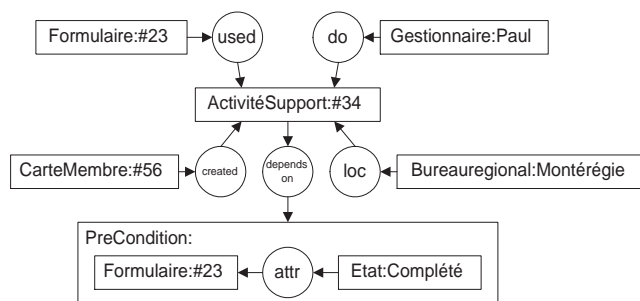


FIG. 4.16: Une demande d'adhésion.

Cette figure nous montre que cette instance spécifique de traitement d'adhésion a été faite par Paul et qu'elle a pris place dans le bureau régional de la Montérégie. Paul a traité un formulaire qui était dans l'état complété et il a créé une carte de membre. Bien que ce genre d'information ne fasse pas partie de la mémoire d'entreprise, nous l'avons représenté avec le modèle uniforme afin de voir le lien avec le modèle qui représente cette activité. Rappelons que le but d'une mémoire d'entreprise n'est pas de traiter des données opérationnelles. Ces dernières sont prises en charge par les systèmes transactionnels de l'entreprise. Quant à la réalisation du processus elle peut être assistée par des outils spécialisés (workflow manager). Toutefois notre modèle permet de lier les données opérationnelles avec la mémoire d'entreprise proprement dite. Avec des méta-informations sur la structure et les moyens d'accès aux données opérationnelles, le lien entre connaissances corporatives génériques et données opérationnelles peut être fait.

Une mémoire d'entreprise conserve les connaissances corporatives, en particulier, la description des processus d'affaires. Les processus d'affaires sont représentés par des modèles. Un modèle décrit d'une manière générique un processus d'affaires. La figure 4.17 présente le modèle<sup>1</sup> de l'activité de traitement d'une demande d'adhésion.

La figure montre les connaissances stockées dans la mémoire d'entreprise au sujet de l'activité Traiter une demande. Nous voyons que cette activité concerne la fonction support. Le concept ou notion de fonction support est décrite ailleurs dans la mémoire d'entreprise. Dans un contexte bilingue, les libellés et les descriptions sont définis en français et en anglais. Le libellé est le nom de l'activité; sa description est un court texte présentant l'activité. L'activité Traiter une demande est définie comme un

<sup>1</sup> basée sur une ontologie du domaine d'application

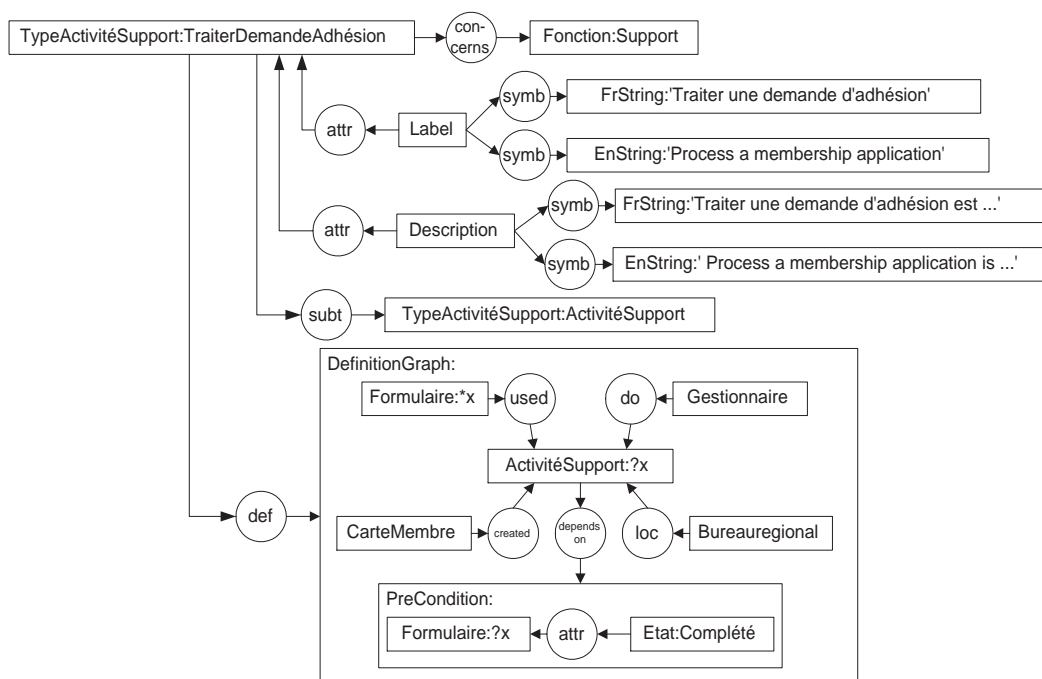


FIG. 4.17: Modèle de l'activité Traiter une demande.

sous-type de `ActivitéSupport`. Puis la figure montre le graphe de définition de l'activité. Ce graphe indique que cette activité est une activité de support utilisant en entrée un formulaire qui doit être complété (comme indiqué par la pré-condition). Elle est exécutée par un gestionnaire, elle prend place dans un bureau régional et une carte de membre est créée comme résultat de l'activité.

Nous venons de voir un exemple de modélisation d'une activité d'affaires de la coopérative. Un autre exemple concernant la coopérative est sa mission. La figure 4.18 montre la représentation de la mission dans la mémoire d'entreprise. La mission de la coopérative est de fournir les meilleurs produits fromagers au meilleur prix.

Dans les deux exemples il s'agit d'information de premier niveau, c'est à dire des informations qui vont supporter directement les employés dans leur tâche, le premier exemple en décrivant une activité, le deuxième en fournissant de l'information corporative. Mais dans les deux cas cette information pourrait s'avérer insuffisante, un employé pourrait avoir besoin de méta-informations. Il pourrait se demander "Qu'est-ce qu'une activité de support?" ou encore "Qu'est-ce qu'une mission?" Les figures 4.19 et 4.20 montrent les connaissances permettant de répondre à ces questions.



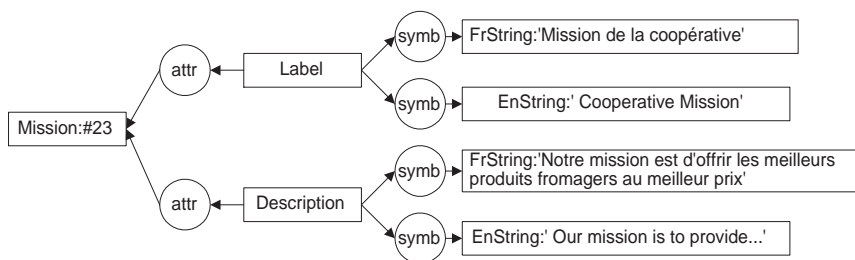


FIG. 4.18: La mission de la coopérative.

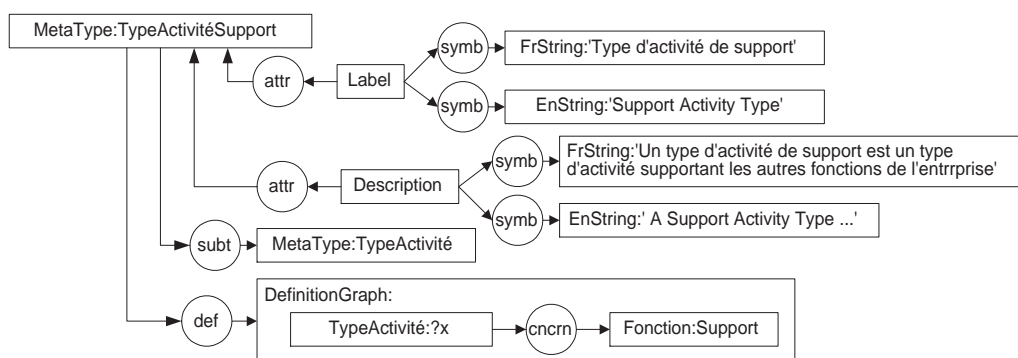


FIG. 4.19: "Qu'est-ce qu'une activité de support ?".

La figure 4.19 présente le graphe décrivant un type d'activité de support. Comme tout concept ou notion de la mémoire d'entreprise, il a un libellé et une description. Nous voyons également que le type d'activité de support est un sous-type de TypeActivité et qu'une instance, c'est à dire, un type d'activité de support concerne la fonction support.

La figure 4.20, quant à elle, présente le graphe décrivant une mission. Le concept a un libellé et une description qui définit la notion de mission. De plus, nous apprenons qu'une mission est une direction qui guide l'entreprise.

Nous terminons cette démonstration par une illustration. La figure 4.21 présente simultanément trois des niveaux de la pyramide des connaissances. Le niveau inférieur concerne les données opérationnelles. Le niveau intermédiaire présente le modèle du processus d'affaire dont le niveau inférieur est une instance. Enfin le niveau supérieur montre les méta-connaissances sur les processus d'affaires.

Ces exemples ont montré, comme nous l'espérons, l'importance de la modélisation et de la métamodélisation d'une mémoire d'entreprise.

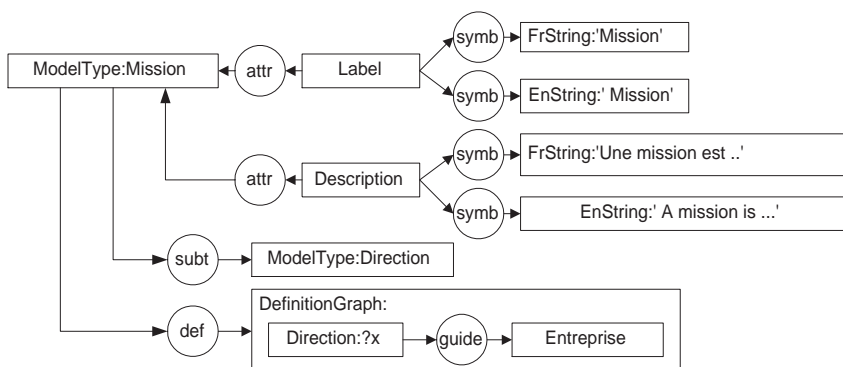


FIG. 4.20: "Qu'est ce qu'une mission ?".

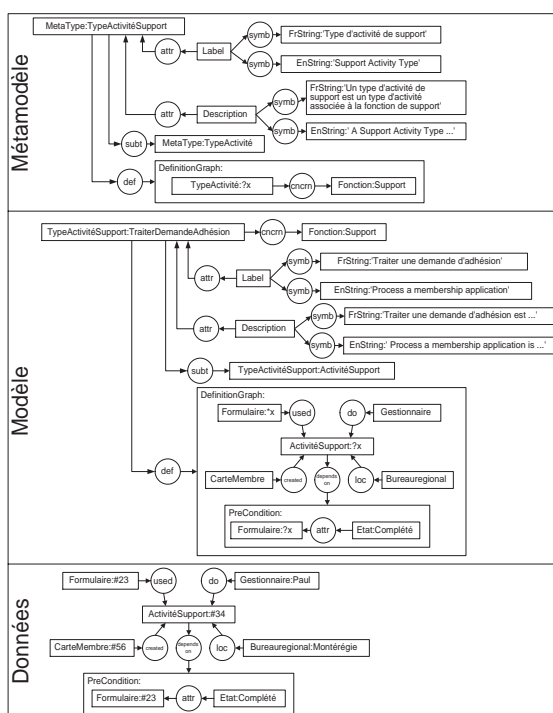


FIG. 4.21: Trois niveaux de la pyramide.

## 4.4 Sommaire

Nous venons de voir dans ce chapitre le méta-métamodèle et les métamodèles de notre proposition comme illustré par la figure 4.22.

Nous avons présenté dans ce chapitre le méta-métamodèle qui définit les éléments

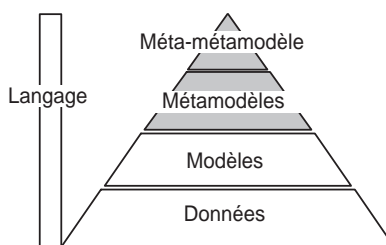


FIG. 4.22: La partie méta-métamodèle et métamodèle de la pyramide des modèles.

permettant de développer les formalismes de représentation. Un formalisme de représentation correspond à des métamodèles. Nous avons présenté ici ceux des graphes conceptuels étendu pour la représentation des connaissances d'une mémoire d'entreprise. Nous avons détaillé le métamodèle de structure, le métamodèle de dynamique, le métamodèle de recherche, et le métamodèle de contrainte.

Nous avons vu le métamodèle de structure pour représenter

- la structure ou l'organisation d'une entreprise (Type de structure),
- les informations manipulées par l'entreprise (Type d'information),
- les objets d'affaires de l'entreprise (Type d'objets d'affaires).

Nous avons vu le métamodèle de dynamique pour représenter

- les différents systèmes qui composent l'univers de l'entreprise (Type de système),
- les objets participant aux systèmes (Type de participant de système),
- les processus d'affaires complexes de l'entreprise (Type de modèle de processus),
- les événements qui marquent la fin d'activités (Type d'événement),
- les processus (Type de processus),
- les participants aux processus d'affaires (Type de participant),
- les états caractérisant la vie des entités de l'entreprise (Type d'états),
- les conditions qui prévalent avant ou après la réalisation des activités (Type de condition).

Enfin nous avons vu les métamodèles de recherche et de contrainte pour rechercher dans la base de connaissance et pour contraindre les connaissances à l'acquisition et tout au long du cycle de vie de la mémoire d'entreprise.

Ce chapitre termine la présentation de notre modèle uniforme. Le chapitre suivant présente les évaluations que nous avons menées sur le modèle uniforme.

# Chapitre 5

## Évaluation du modèle uniforme

Le modèle développé dans cette thèse répond aux besoins théoriques et pratiques. D'autres systèmes peuvent également répondre à ces besoins. Mais notre modèle le fait dans un cadre intégré ; c'est le seul, à notre connaissance, qui a une approche de définition globale.

Les objectifs de notre travail de recherche étaient de montrer : 1) que les graphes conceptuels permettent de définir un modèle uniforme de représentation de la connaissance et 2) que ce modèle uniforme permet de modéliser tous les types de connaissances nécessaires à la constitution d'une mémoire d'entreprise.

L'atteinte de l'objectif 1 est vérifiée. La définition du modèle uniforme aux chapitres 3 et 4 montre que les graphes conceptuels permettent de définir les différents niveaux de modélisation.

L'évaluation de l'objectif 2 a été faite en deux parties : une partie théorique qui montre que chacun des types de connaissances d'une mémoire d'entreprise peut être représenté par le modèle uniforme, et une partie pratique qui montre sur un exemple concret la modélisation d'une mémoire d'entreprise.

### 5.1 Évaluation théorique

L'évaluation théorique s'est faite en cinq étapes :

- définition et modélisation des besoins concernant la partie statique de la connaissance,
- définition et modélisation des besoins concernant la partie dynamique de la connaissance,

- définition et modélisation des besoins concernant les contraintes,
- définition et modélisation des besoins concernant la recherche,
- confrontation du modèle avec les modèles d’expertise de CommonKads [77].

### 5.1.1 Besoins concernant la partie statique de la connaissance

Les besoins spécifiques d’une mémoire d’entreprise concernant la partie statique de la connaissance ont été spécifiés pendant les travaux de modélisation des méthodologies du *Groupe Conseil DMR Inc.* Ces besoins sont :

- classification et connaissance partielle,
- relations entre catégories et /ou instances,
- catégorie ou instance et métamodèle,
- relations et cardinalités.

Les trois premiers ont été présentés en détail dans le chapitre 2, nous les reprenons donc ici d’une manière succincte. Pour chacun de ces types de besoins, nous montrons comment le modèle proposé permet de combler ces besoins.

#### Classification et connaissance partielle

Pour répondre à ce besoin le formalisme doit supporter la multi-classification, c’est-à-dire qu’un objet peut être instance de plus d’une catégorie, ou le système doit être capable de faire migrer dynamiquement les instances d’une catégorie à une autre catégorie.

Le modèle supporte les deux mécanismes. Un même objet peut être multi-classifié. Ceci signifie que le référent représentant l’objet peut être associé à des types différents dans des concepts différents. La figure 5.1 illustre cette propriété. Le même département de support technique aux employés représenté dans le système par le référent #9761 peut être interprété comme un centre de coût par le département des finances et comme un service par les autres départements.



FIG. 5.1: Multi-classification.

Nous avons vu lors de la définition du langage (chapitre 3) du modèle uniforme qu’un référent peut être associé à un type si le concept formé par cette association

est conforme à la définition du type, c'est-à-dire, qu'il a les relations conceptuelles présentées dans le graphe de définition, et que, de plus, il respecte les contraintes exprimées par les graphes de restriction ou de règle associés au type. La figure 5.2 montre sur un exemple comment, grâce à ce mécanisme, un concept peut migrer d'une catégorie à une autre.

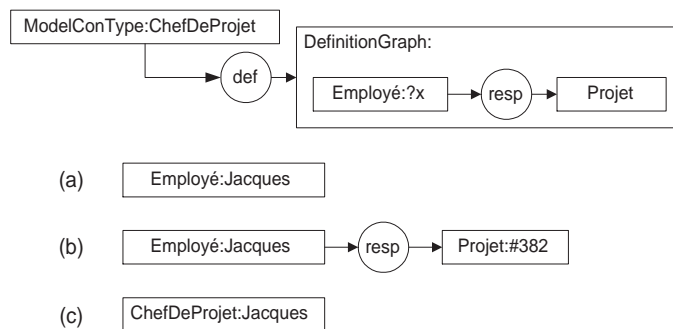


FIG. 5.2: Multi-classification, migration et connaissance partielle.

Le premier graphe montre la définition du type ChefDeProjet. Un chef de projet est un employé responsable (*resp*) d'un projet. Le graphe (a) indique que Jacques est un employé. L'ajout de la relation responsable du projet #382 dans le graphe (b) fait que le concept [Employé :Jacques] vérifie la définition du type ChefDeProjet. Nous pouvons donc écrire que Jacques est un chef de projet (c).

Ce mécanisme permet de répondre aux problèmes de la connaissance partielle. Si au moment de la saisie d'une nouvelle connaissance, nous ne connaissons pas toutes les informations, nous ne renseignerons que celles qui sont connues. Par la suite l'ajout d'informations complémentaires fera migrer l'objet vers des types plus spécialisés.

### Relations entre catégories et/ou instances

Pour répondre à ce besoin le formalisme doit supporter les relations entre catégories et/ou instances, c'est-à-dire qu'une catégorie peut être liée à une instance, ou que les relations sont établies au niveau instance.

Le modèle uniforme permet d'exprimer des connaissances au niveau données (instances) et au niveau méta-données. Nous pouvons donc répondre au besoin en définissant les relations au niveau instance. La figure 5.3 montre sur un exemple comment

une instance, ici la ville de Montréal, peut être utilisée dans la définition d'un type<sup>1</sup>.

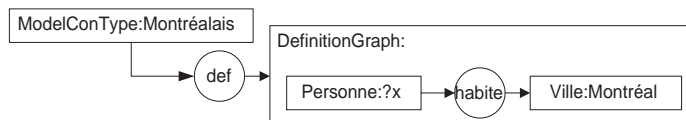


FIG. 5.3: Relations définies au niveau instance.

### Catégorie ou instance et métamodèle

Pour répondre à ce besoin le formalisme doit permettre qu'un élément soit vu comme une catégorie ou comme une instance dépendant du point de vue.

Le modèle uniforme est un langage de modélisation et de métamodélisation. Il est particulièrement bien adapté pour ce besoin. Les figures 5.4 et 5.5 montrent la notion de Mission respectivement comme un type dont on présente une instance et comme une instance dont on donne la définition.



FIG. 5.4: Mission vue comme un type.

La figure 5.4 montre le graphe représentant la mission d'une entreprise d'outillage. La mission de l'entreprise est symbolisée par une chaîne de caractères exprimant la mission de l'entreprise. Ce graphe permet dans le contexte d'une mémoire d'entreprise de répondre à la question : "Quelle est la mission de l'entreprise?". Par contre, elle ne permet pas de répondre à "Qu'est-ce qu'une mission?"

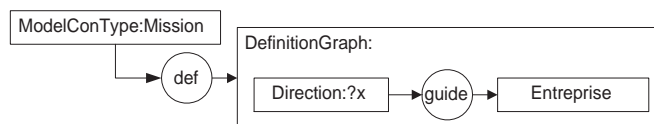


FIG. 5.5: Mission vue comme une instance.

<sup>1</sup> Un ModelConType est un type de concept de niveau modèle (section 4.2)



La figure 5.5 montre le graphe représentant la définition du type Mission, c'est-à-dire, représentant ce qu'est une mission. Ce graphe permet de répondre à la question "Qu'est-ce qu'une mission?".

## Relations et cardinalités

Il existe une notion importante dans les formalismes de modélisation, la notion de cardinalité d'une relation. Une relation permet d'associer des éléments entre eux. Les cardinalités permettent de spécifier le nombre d'éléments qui peuvent être impliqués. Cette notion permet de répondre au besoin d'exprimer comment les éléments peuvent être associés. Nous présentons ci-dessous les différentes valeurs possibles pour les cardinalités et nous montrons comment exprimer cette information avec le modèle uniforme.

**Cardinalités 1..1** La cardinalité 1..1 correspond à une relation unique et obligatoire. La figure 5.6 illustre la cardinalité 1-1. Dans cet exemple, nous exprimons la règle qui dit qu'un employé a un et un seul numéro d'employé. Cette cardinalité est définie en deux temps. Dans un premier temps, le graphe de définition spécifie qu'un employé est une personne qui a un numéro d'employé. La contrainte définie ici est le minimum. Dans un deuxième temps le graphe de restriction spécifie qu'un employé ne peut pas avoir deux numéros d'employés ; la contrainte définie ici est le maximum : un numéro.

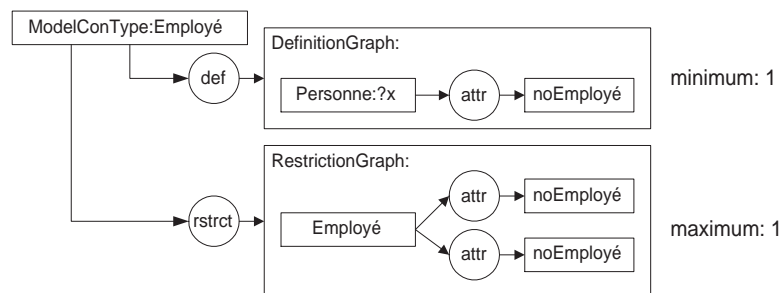


FIG. 5.6: Cardinalités : minimum 1, maximum 1.

**Cardinalités 1..n** La cardinalité 1..n correspond à une relation obligatoire mais possiblement multiple. La figure 5.7 illustre la cardinalité 1-n. Dans l'exemple, nous

exprimons que pour être chef de projet, il faut être responsable d'un projet, mais que l'on peut être responsable de plusieurs projets. La seule contrainte est donc le minimum qui est spécifié dans le graphe de définition.

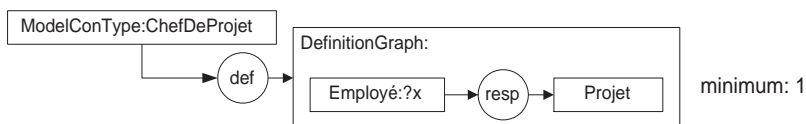


FIG. 5.7: Cardinalités : minimum 1, pas de maximum.

**Cardinalités  $i..j$**  La cardinalité  $i..j$  avec  $i > 0$  et  $i \leq j$  correspond à une relation dont les minima et maxima sont déterminés. La figure 5.8 illustre la cardinalité  $2..3$ . Une personne a un et un seul nom et a deux ou trois prénoms. Les cardinalités minimales sont exprimées dans le graphe de définition et les cardinalités maximales sont exprimées dans les graphes de restriction.

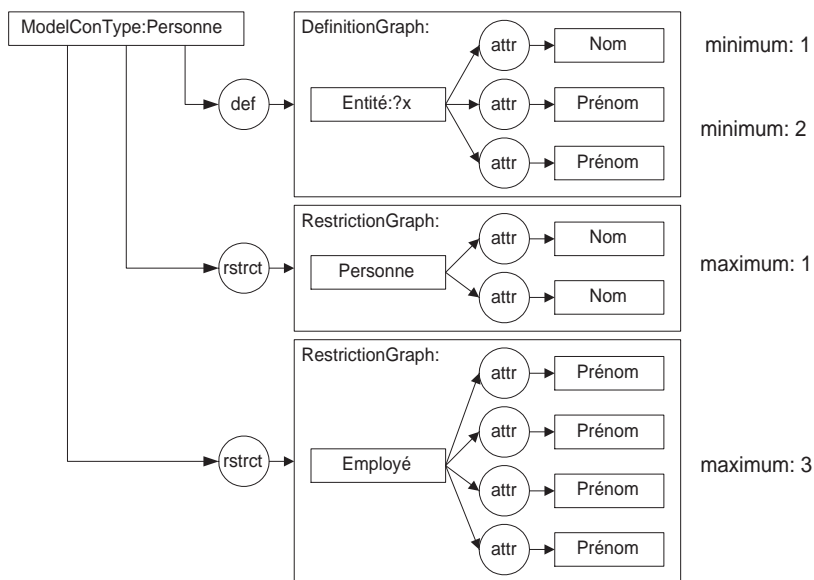


FIG. 5.8: Cardinalités : minimum  $i$ , maximum  $j$ .

**Cardinalités  $0..x$**  La cardinalité  $0..x$  avec  $x > 0$  correspond à une relation optionnelle. Comme la relation ne peut pas apparaître dans le graphe de définition, nous

devons, pour permettre son utilisation, définir un sous-type pour lequel la cardinalité de la relation est 1..x. La figure 5.9 illustre le cas de la relation possède *poss*. Une personne possède des téléphones. La relation est optionnelle. Nous distinguons alors un sous-type de personne *PersonnePossTel* pour lesquels la cardinalité est 1..n.

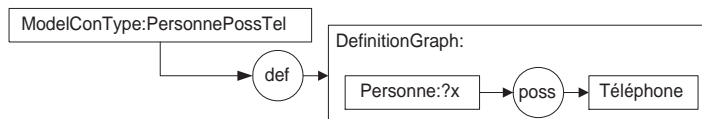


FIG. 5.9: Sous type avec cardinalité 1..n.

En conclusion, les minima des cardinalités sont exprimés dans les graphes de définition des types concernés. Les maxima sont spécifiés dans les graphes de restriction. Dans le cas où le minimum est zéro, on distingue le sous-type pour lequel la cardinalité a pour minimum 1. Le modèle uniforme permet donc de représenter tous les besoins liés aux cardinalités. Il est à noter qu'il faut différencier la représentation conceptuelle que nous venons de voir du mécanisme d'acquisition qui pourrait être différent pour faciliter l'acquisition des cardinalités. Un mécanisme d'acquisition pourrait utiliser une notation plus classique comme les cardinalités sur les arcs.

### 5.1.2 Besoins concernant la partie dynamique de la connaissance

Dans le chapitre 2 nous avons défini des besoins spécifiques pour la partie dynamique de la connaissance à partir des travaux de modélisation des processus des méthodologies du *Groupe Conseil DMR Inc.* Nous reprenons dans cette section chacun de ces besoins et nous montrons comment le modèle uniforme nous permet de proposer une réponse à ces besoins.

#### Partage d'activités

Pour supporter les besoins de représentation de la dynamique, le formalisme doit supporter la représentation de processus partageant les mêmes activités.

Le modèle permet de représenter le partage d'activités. La figure 5.10 reprend l'exemple du chapitre 2 et montre la définition de deux processus concourant au développement d'un produit composé d'une partie matérielle et d'une partie logicielle.

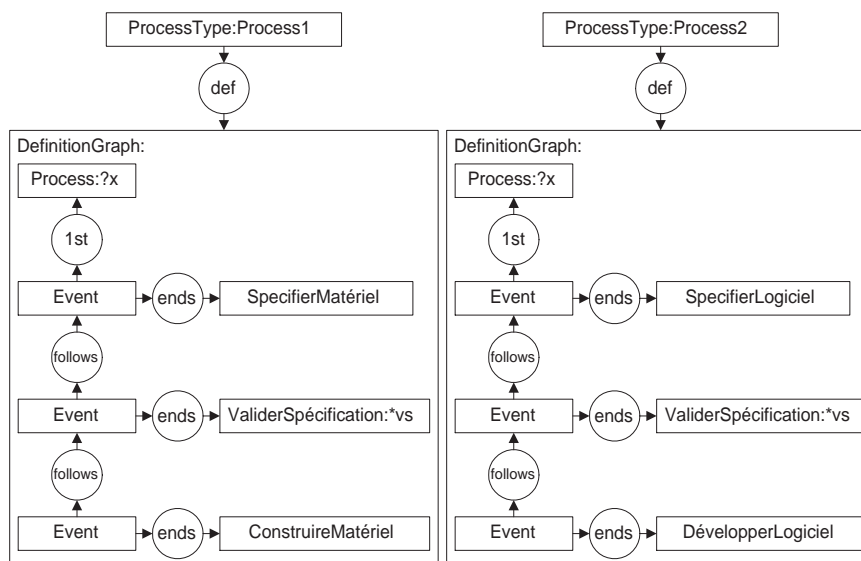


FIG. 5.10: Partage d'activités.

Le premier processus `Process1` concerne l'équipe de développement du matériel. Le deuxième processus `Process2` concerne l'équipe de développement du logiciel. Les deux processus ont une activité commune qui permet de valider les spécifications matérielles et logicielles. L'utilisation du même nom de variable indique que les deux concepts sont dans un ensemble de co-référence et donc qu'ils représentent la même activité.

### Gestion des instances

Pour supporter les besoins de représentation de la dynamique, le formalisme doit supporter la représentation et la gestion des instances dans les processus.

Le formalisme des graphes conceptuels permet d'exprimer les connaissances au niveau des instances, il est donc possible de gérer les instances dans les processus. La figure 5.11 illustre un problème similaire à celui de la fenêtre vu au chapitre 2. Il s'agit de décrire le processus d'une entreprise de confection de costumes sur mesure. À partir des mensurations du client, un patron du costume est dessiné. Le pantalon et la veste sont alors fabriqués en parallèle, puis regroupés pour essayage. Le problème est d'assembler le pantalon avec la bonne veste.

Grâce aux ensembles de co-référence, le patron défini dans l'activité `DessinerPatron` est le même que celui utilisé dans les activités `FabriquerVeste`, `FabriquerPantalon` et Li-

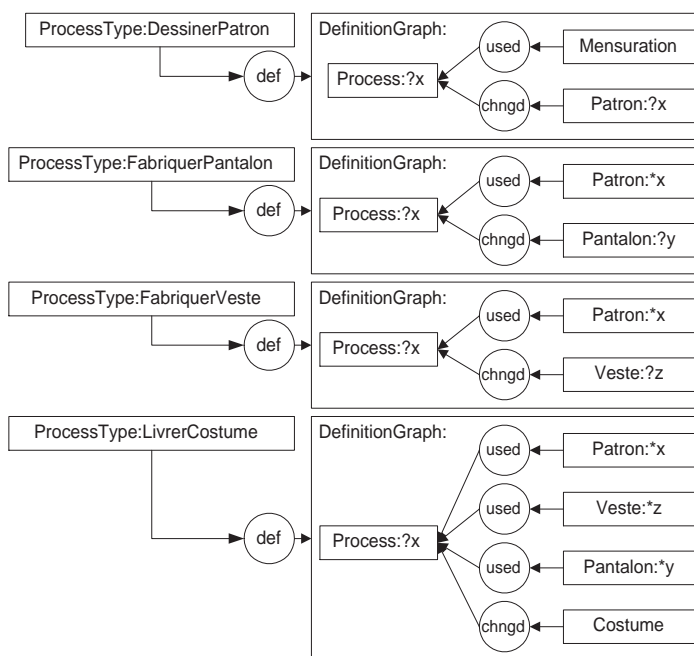


FIG. 5.11: Partage d'activités.

vrerCostume. De même, le pantalon défini dans l'activité FabriquerPantalon est le même que celui utilisé dans l'activité LivrerCostume et la veste définie dans l'activité FabriquerVeste est la même que celle utilisée dans l'activité LivrerCostume. Le costume sera donc bien conforme aux mensurations.

### 5.1.3 Besoins concernant les contraintes

Toute représentation de connaissances nécessite la représentation de contraintes, c'est-à-dire, de règles contraignant les connaissances. Afin de valider le pouvoir d'expression de notre modèle, nous avons pris pour base d'évaluation la liste des contraintes identifiées dans [53]. Pour chacune de ces contraintes nous montrons comment représenter la contrainte avec le modèle uniforme.

#### Contrainte de clé unique

*Le numéro d'assurance social d'un(e) employé(e) est unique.*

Ceci signifie qu'un employé a un et un seul NAS, et un NAS est assigné à un et un seul employé. Cette contrainte est double. Nous avons une fonction d'employé vers

NAS et une fonction de NAS vers employé. Ces deux contraintes sur Employé et sur NAS se retrouvent dans les deux définitions de type comme illustré par la figure 5.12 qui présente les graphes de spécifications de Employé et NAS.

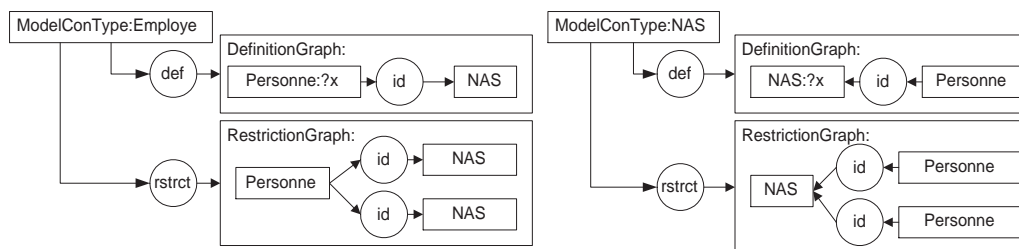


FIG. 5.12: Contrainte de clé unique.

### Contrainte de non nullité

*Le numéro d'assurance social d'un(e) employé(e) doit être obligatoirement renseigné.*

Ceci signifie qu'un(e) employé(e) ne peut pas être ajouté(e) si son NAS n'est pas défini. Cette contrainte signifie que le concept représentant le numéro d'assurance social doit être un concept individuel, c'est-à-dire, un concept dont l'objet qu'il représente est identifié. La figure 5.13 présente les graphes de spécifications de Employé. Le graphe de règle présente une assertion du second ordre.

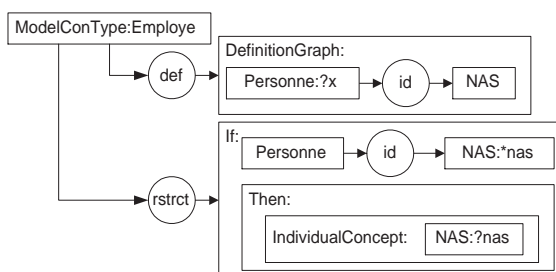


FIG. 5.13: Contrainte de non nullité.

### Contrainte d'appartenance à un domaine

*L'âge d'une personne est inférieur à 130.*

La contrainte est spécifiée dans le graphe de définition du type L'âge d'une personne est un entier inférieur à l'entier 130. La figure 5.14 présente les graphes de spécifications du type AgePersonne.

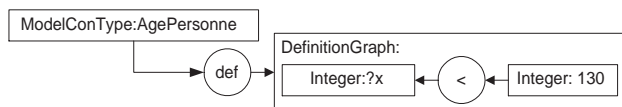


FIG. 5.14: Contrainte d'appartenance à un domaine.

### Contrainte de dépendance fonctionnelle

*Le poste d'un(e) employé(e) détermine son salaire.*

Cette contrainte est une dépendance fonctionnelle entre deux concepts. Il y a dépendance fonctionnelle entre deux concepts si la connaissance du premier concept implique la connaissance du deuxième concept. Autrement dit le premier concept ne peut pas être associé à deux concepts différents.

La figure 5.15 présente les graphes de spécifications du type Employé. Le graphe de restriction implémente la contrainte.

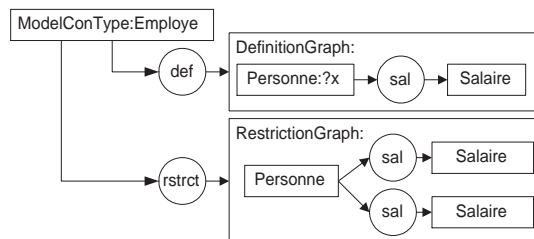


FIG. 5.15: Contrainte de dépendance fonctionnelle.

Les dépendances fonctionnelles sont des éléments fondamentaux de la théorie de la normalisation du modèle relationnel ; elles s'expriment facilement par une contrainte sur la cardinalité des relations. Il existe une dépendance fonctionnelle du concept A vers le concept B si la cardinalité de la relation associant A à B a pour cardinalité maximale 1.

### Contrainte de dépendance fonctionnelle inter relation

*Si un(e) employé(e) dirige un département, alors il ou elle doit travailler dans ce département.*

Ceci représente le cas où un graphe implique un autre graphe qui est une spécialisation du premier. C'est un type de dépendance fonctionnelle. La figure 5.16 présente les graphes de spécifications du type Directeur. Le graphe de règle associé au type implémente la contrainte.

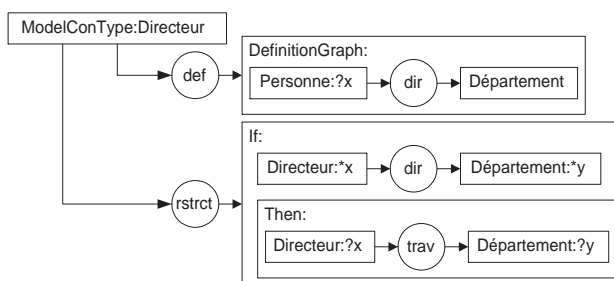


FIG. 5.16: Contrainte de dépendance fonctionnelle inter relation.

### Contrainte de dépendance d'inclusion

*Tout(e) employé(e) travaille dans un département existant.*

Dire qu'un département est existant s'exprime en disant que le concept représentant le département est un concept individuel. Un concept individuel est un concept représentant un élément connu et identifié. La figure 5.17 présente les graphes de spécifications du type Employé. Le graphe de règle présente une assertion du second ordre. Si un employé travaille dans un département alors le concept représentant ce département est un concept individuel.

### Contrainte d'implication

*Si le poste d'un(e) employé(e) est gérant, alors son salaire est supérieur à \$50,000.*

Une contrainte d'implication est représentée par un graphe de règle. La figure 5.18 présente les graphes de spécification du type Gérant. Le graphe de règle implémente la contrainte.



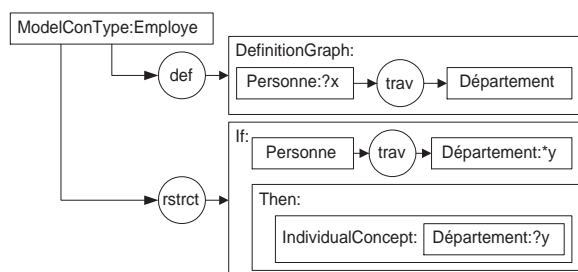


FIG. 5.17: Contrainte de dépendance d'inclusion.

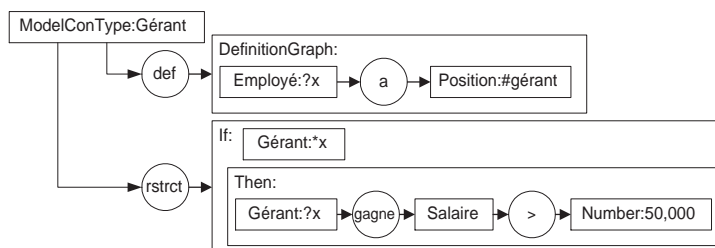


FIG. 5.18: Contrainte d'implication.

### Contrainte de migration

*Un étudiant sous-gradué devient gradué et non l'inverse.*

Une contrainte de migration est une contrainte liée à l'évolution des données et donc à la prise en considération du temps. Trois solutions sont possibles pour exprimer cette contrainte. Une première solution est statique et laisse à l'utilisateur le contrôle de l'évolution. La contrainte s'exprime dans ce cas par un graphe de règle dans la définition du type *Gradué*. Si un étudiant est gradué alors il n'est pas sous-gradué. La figure 5.19 présente cette solution.

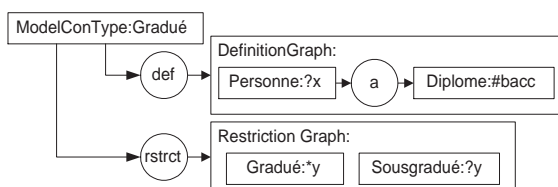


FIG. 5.19: Contrainte de migration.

Une deuxième solution est d'exprimer la contrainte par une assertion du deuxième ordre qui associe les types `Gradué` et `SousGradué` par une relation de migration. La figure 5.20 montre le graphe de deuxième ordre implémentant la contrainte.



FIG. 5.20: Contrainte de migration.

Cette deuxième solution implique la mise en place d'un mécanisme vérifiant la classification des entités.

Une troisième solution est de spécifier les contraintes de migration par l'utilisation des pré-conditions et post-conditions des activités. La figure 5.21 illustre l'exemple de la modification d'un salaire où le salaire ne peut pas baisser. La post-condition précise que le salaire après modification est associé à un nombre qui doit être strictement supérieur à celui qui était lié avant la modification.

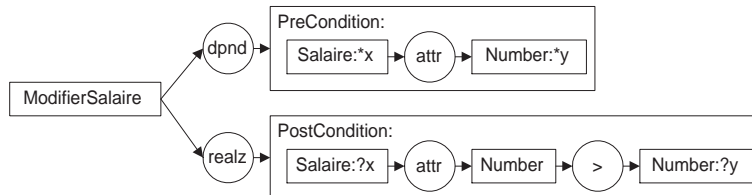


FIG. 5.21: Contrainte de migration.

## Contrainte de couverture

*Un étudiant est sous-gradué ou gradué.*

La contrainte de couverture exprime le fait que deux sous-types ou plus représentent toutes les alternatives pour le type parent. Dans l'exemple un étudiant ne peut pas être autre chose que gradué ou sous-gradué. La figure 5.22 présente un graphe de règle illustrant cette contrainte.

Montrons que la clause 'Si A alors B ou C' est bien équivalente à deux restrictions imbriquées dans une troisième. Les lignes suivantes sont équivalentes :

$$A \Rightarrow (B \vee C)$$

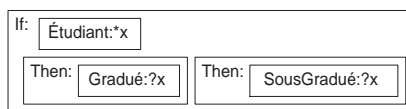


FIG. 5.22: Contrainte de couverture.

$$\begin{aligned}
 & \neg A \vee (B \vee C) \\
 & \neg (A \wedge \neg (B \vee C)) \\
 & \neg (A \wedge \neg B \wedge \neg C) \\
 & \neg [\text{Graph} : A \neg [\text{Graph} : B] \neg [\text{Graph} : C] ] \\
 & [\text{RestrictionGraph} : A [\text{RestrictionGraph} : B] [\text{RestrictionGraph} : C] ] \\
 & [\text{If} : A [\text{Then} : B] [\text{Then} : C] ]
 \end{aligned}$$

Il est à remarquer que le "ou" n'est pas exclusif et donc qu'il ne s'agit pas de la clause "Si A alors B sinon C" qui implique une exclusion mutuelle entre B et C.

### Contrainte de disjonction

*Les types Gradué et SousGradué sont disjoints.*

Ceci signifie qu'il n'y a pas d'éléments communs aux deux types. Un gradué ne peut pas être sous-gradué et réciproquement. La figure 5.23 présente les graphes de spécifications du type SousGradué. Le graphe de définition nous montre qu'un sous-gradué est une personne ayant un diplôme d'études collégiales (DEC). Le graphe de restriction nous montre qu'un sous-gradué ne peut pas être gradué.

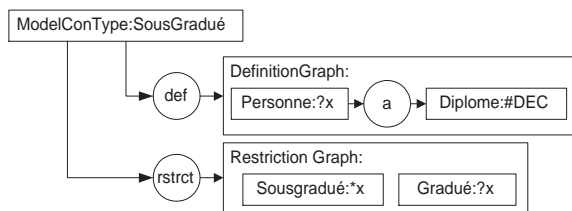


FIG. 5.23: Contrainte de disjonction.

En combinant cette définition avec celle du type Gradué présentée à la figure 5.19 nous avons ainsi la représentation de la contrainte de disjonction.

### Contrainte de contexte

*Un(e) étudiant(e) est payé(e) avec les fonds de recherche d'un(e) professeur(e) subventionné(e) qui le ou la supervise.*

Il s'agit d'une dépendance fonctionnelle entre deux graphes. La figure 5.24 présente le graphe de règle exprimant la contrainte.

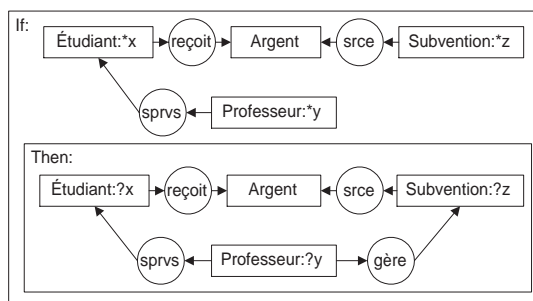


FIG. 5.24: Contrainte de contexte.

Si un(e) étudiant(e) est supervisé(e) par un(e) professeur(e) et reçoit de l'argent provenant d'une subvention alors le ou la professeur(e) gère cette subvention.

### Contrainte d'implication d'informations négatives

*Il n'y a pas de salle de réunion au 9<sup>ème</sup> étage.*

La représentation de la contrainte se fait par l'utilisation d'un graphe de restriction qui nie la présence d'une salle de réunion à l'étage numéro 9. La figure 5.25 montre le graphe de restriction correspondant à la contrainte.

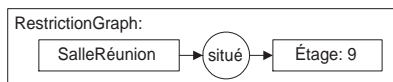


FIG. 5.25: Contrainte d'implication d'informations négatives.

### Contrainte de restriction de domaine

*Le chercheur #1234 n'a, comme subvention, que la subvention du CRSNG.*

Cette contrainte est liée à l'individu #1234 de type chercheur. Elle exprime le fait que le chercheur détermine fonctionnellement la subvention. La figure 5.26 présente le graphe de restriction qui montre que le chercheur #1234 ne peut être associé à aucune autre subvention que celle du CRSNG.

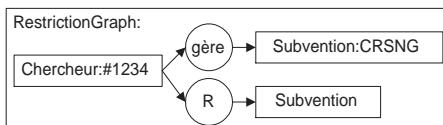


FIG. 5.26: Contrainte de restriction de domaine.

### Contrainte d'exclusivité

*Aucun(e) professeur(e) peut être un(e) étudiant(e) d'un cours qu'il ou elle dispense.*

Ce type de contrainte s'exprime, comme pour les contraintes d'implication d'informations négatives, par un graphe de restriction. La figure 5.27 présente le graphe de restriction empêchant un(e) professeur(e) de suivre son propre cours.

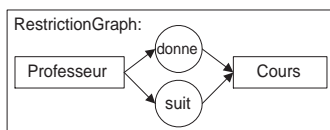


FIG. 5.27: Contrainte d'exclusivité.

### Contrainte d'exclusion

Les contraintes d'exclusion sont des contraintes permettant d'exprimer des alternatives d'associations. Soit, par exemple, la spécification d'un compte bancaire qui précise que le ou les titulaires du compte sont des particuliers ou des entreprises. Mais le mélange des deux n'est pas autorisé. La figure 5.28 illustre comment représenter cette contrainte dans le modèle uniforme.

Nous définissons le type titulaire de compte qui se spécialise en deux types : particulier et entreprise. Puis nous définissons un compte bancaire comme une entité ayant

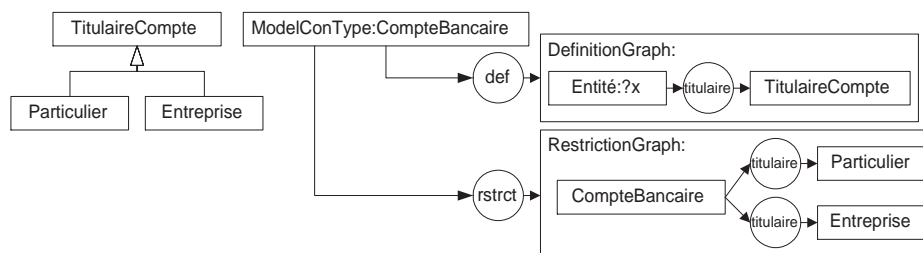


FIG. 5.28: Contrainte d'exclusivité.

pour titulaire un titulaire de compte (particulier et/ou entreprise) et avec un graphe de restriction qui montre qu'un compte bancaire ne peut pas avoir pour titulaire un particulier et une entreprise.

#### 5.1.4 Besoins concernant la recherche

La recherche dans un corpus de connaissance est une fonctionnalité fondamentale. Il serait inutile de stocker des volumes importants de connaissances sans des outils puissants de recherche. Nous avons identifié trois besoins fondamentaux : la recherche d'un graphe dans la base de connaissances, la recherche du type d'un concept et la recherche des instances d'un type. Ces deux derniers besoins permettent le changement de niveau de modélisation.

#### Recherche d'un graphe

La fonction  $\sigma$  présentée à la section 4.2.5 (page 111) est le mécanisme général de recherche dans la base de connaissances. Nous rappelons sa définition (équation 5.1).  $G$  est l'espace de recherche,  $G$  est l'ensemble des graphes sur lequel le graphe de recherche  $q$  va être évalué. L'espace de solution  $\sigma(G, q)$  est l'ensemble des projections de  $q$  dans  $G$ .

$$\sigma(G, q) = \{g \in G \mid g = \pi(q)\} \quad (5.1)$$

Nous avons vu à la section 4.2.5 (page 112) que la fonction  $\sigma$  permettait de naviguer dans les contextes et d'y rechercher des graphes. Nous ne reprendrons donc pas ici la démonstration.

Le langage défini dans cette thèse permet de représenter d'une manière uniforme les différents niveaux de modélisation. Le mécanisme de recherche permet à partir d'un niveau de modélisation de rechercher dans le niveau supérieur, il s'agit de la recherche du type d'un concept, ou de rechercher dans un niveau inférieur, il s'agit de la recherche des concepts d'un type.

### Recherche du type d'un concept

En se basant sur la fonction  $\sigma$ , nous montrons comment il est possible de rechercher le type d'un concept donné. Soit le concept `[ChefDeProjet :#234]` représentant un chef de projet, alors le graphe de recherche ci-dessous permet d'accéder au type `ChefDeProjet`.

$$q_1 : [\text{Concept} : [\text{ChefDeProjet} :\#234] ] \rightarrow (\text{type}) \rightarrow [\text{ConceptType} :*\text{t}]$$

### Recherche des concepts d'un type

Le même mécanisme nous permet de passer à un niveau inférieur de modélisation. Soit le type `ChefDeProjet`, alors le graphe de recherche ci-dessous permet de retrouver tous les concepts de type `ChefDeProjet`.

$$q_2 : [\text{ChefDeProjet} :*\text{p}]$$

Le mécanisme de recherche, basé sur la fonction  $\sigma$ , est général. Il permet de rechercher dans tous les niveaux de modélisation mais aussi de rechercher dans un autre niveau de modélisation. Associé à la fonction  $\omega$ , ce mécanisme permet de se projeter sur un autre niveau. Il est ainsi possible d'avoir une seule base de connaissances mais stratifiée par niveau de modélisation.

#### 5.1.5 Confrontation avec CommonKads

La dernière étape de notre évaluation théorique concerne la confrontation avec les modèles d'expertise définis dans CommonKads. CommonKads est une méthodologie pour l'ingénierie de systèmes à base de connaissances. CommonKads définit des modèles d'expertise [77] qui représentent les diverses connaissances nécessaires à la réalisation d'une application de type système expert. Notre confrontation s'est faite sur la base de CML (Conceptual Modelling Language) [65], un langage et une notation structurés et semi-formels pour les modèles d'expertise. Le langage CML couvre

les connaissances liées au domaine (incluant la spécification des ontologies), liées à l'inférence, liées aux tâches ainsi qu'aux méthodes de résolution de problèmes. Nous montrons ci-dessous comment le modèle uniforme proposé dans cette thèse couvre les modèles d'expertise de CommonKads. Mais avant nous présentons les principales caractéristiques de CommonKads.

Dans les années 80, la communauté de l'intelligence artificielle commence à prendre conscience du manque de méthode pour développer des systèmes à bases de connaissance. Deux projets européens ESPRIT (P1098 et P5248) vont combler ce manque. De 1983 à nos jours ces projets vont donner naissance à KADS puis CommonKADS [65, 66, 78].

Cette section est organisée comme suit. Nous présentons d'abord les différents modèles développés dans la méthode CommonKADS. Puis nous détaillons le modèle d'expertise où les efforts de spécification ont été les plus importants et présentons ensuite le langage de modélisation développé pour représenter le modèle d'expertise.

## Les modèles de CommonKADS

La méthode CommonKADS a été développée pour supporter la réalisation de systèmes experts. Un système expert, comme tout autre système d'information, doit s'intégrer dans l'organisation au point de vue culture, structure et processus. Pour cela la méthode CommonKADS définit un ensemble de modèles permettant de représenter le contexte dans lequel doit s'intégrer le système expert. La méthode définit six modèles présentant les différentes facettes du développement d'un système expert. La Figure 1 montre les six modèles ainsi que leurs relations.

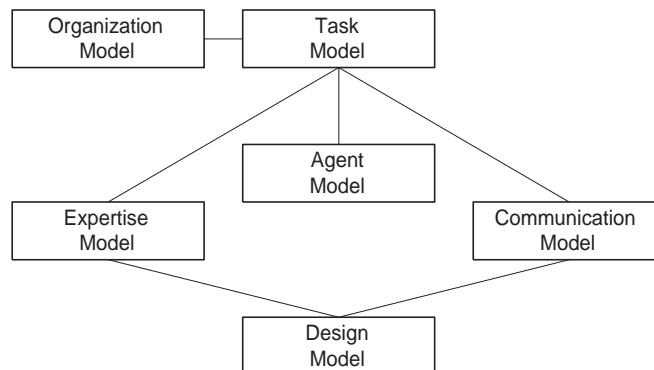


FIG. 5.29: Les modèles de CommonKads.



*Le modèle de l'organisation (Organization Model).* Le modèle de l'organisation décrit le contexte organisationnel dans lequel vient s'intégrer le développement du système expert. Il sert à l'analyse et à la découverte des problèmes et des opportunités pour le développement d'un système expert.

*Le modèle de tâches (Task Model).* Le modèle de tâches décrit les tâches impliquées par le développement du système expert. Il décrit les activités de l'organisation dans le contexte où on envisage d'implanter un système expert. Le modèle de tâches spécifie les tâches et leur décomposition ainsi que les intrants, extrants et exécutants. Le modèle de tâches est un modèle "business" ; il décrit le processus dans lequel s'inscrit le système expert et non le processus de raisonnement de l'expert.

*Le modèle des agents (Agent Model).* Le modèle d'agent décrit les intervenants avec leurs rôles et responsabilités. Il décrit les caractéristiques et capacités des agents. Le modèle permet également de spécifier les contraintes applicables aux agents, comme, par exemple, les normes, préférences ou encore les permissions des agents.

*Le modèle de communication (Communication Model).* Le modèle de communication décrit les communications entre agents. Il décrit les interactions possibles entre les agents. Le modèle représente les communications indépendamment d'une possible automatisation.

*Le modèle d'expertise (Expertise Model).* Le modèle d'expertise décrit les connaissances de l'expert que le système expert doit implémenter. Le modèle d'expertise est le modèle fondamental de la méthode KADS. Le modèle d'expertise décrit le comportement d'un agent en termes des raisonnements que cet agent doit effectuer pour réaliser la tâche assignée.

*Le modèle de design (Design Model).* Le modèle de design (Design Model) décrit la structure du système expert à développer. Il décrit les structures et les mécanismes à mettre en place pour construire le système expert. Le modèle est constitué (i) d'une architecture qui définit les composantes de base qui vont permettre de construire le système, (ii) d'un design d'application qui fait la correspondance entre le modèle d'expertise, le modèle de communication et les composantes de base de l'architecture, et (iii) d'un design de la plate-forme logicielle et matérielle pour implanter le système expert.

Le modèle d'expertise est le modèle fondamental de CommonKADS. Ce modèle d'expertise est détaillé dans la section suivante.

## Le modèle d'expertise

Le modèle d'expertise permet de représenter la connaissance du domaine, la connaissance de raisonnement et la connaissance sur la décomposition des problèmes pour les résoudre.

**Connaissance du domaine (Domain Knowledge).** La description de la connaissance du domaine définit le contenu du domaine ainsi que sa structure. La structure définit les types d'objet du domaine ainsi que les relations que ces objets peuvent entretenir. Cette définition de structure se fait au travers de la définition d'une ontologie du domaine étudié. La figure 5.30 présente les différentes composantes de "domain knowledge" avec les relations entre ces composantes. Les modèles de domaine décrivent les différents points de vue possibles sur le domaine. L'ontologie du domaine fournit le vocabulaire nécessaire à la description des points de vue. L'ontologie du modèle est une "méta description" des éléments du domaine. Cette ontologie fournit les termes nécessaires à la description des structures des éléments du domaine.

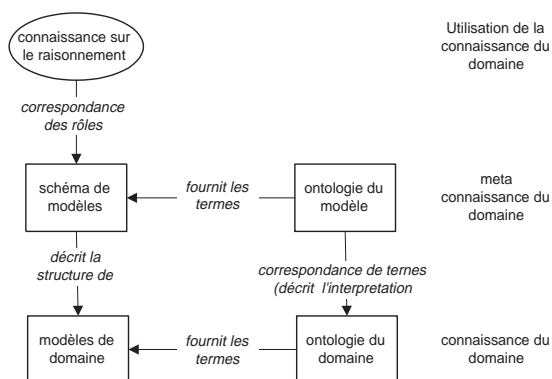


FIG. 5.30: Relations entre les composantes de "domain knowledge".

**Connaissance sur le raisonnement (Inference Knowledge).** La connaissance sur le raisonnement est décrite en termes d'opérations ou d'inférences sur les éléments du domaine et en termes de rôles. Dans ce modèle sont décrits les raisonnements de base qui, une fois ordonnés, vont former les tâches.

**Connaissance sur les tâches (Task Knowledge).** La connaissance sur les tâches est modélisée comme une hiérarchie de tâches. La définition d'une tâche est composée de deux parties : une partie qui décrit le but de la tâche, c'est-à-dire ce qui doit être fait et une partie qui décrit la procédure à suivre, c'est-à-dire comment cela doit être fait.

### **Le langage de modélisation conceptuelle (CML)**

Le langage de modélisation conceptuelle (Conceptual Modelling Language) permet de modéliser et spécifier le modèle d'expertise. Le langage fournit une notation textuelle et une notation graphique proche de celle d'UML. Pour chaque composante du modèle d'expertise CML définit les éléments nécessaires à sa modélisation.

**Connaissance du domaine.** La connaissance sur le domaine est constituée de trois éléments : les définitions des ontologies, les correspondances entre ontologies et les modèles de domaines qui définissent les connaissances de la base de connaissances.

*Définition des ontologies.* Une ontologie est définie comme un ensemble de types parmi concept, attribut, expression, structure, et relation. Chacun des types de l'ontologie, concept, attribut, expression, structure, et relation peut être caractérisé par une terminologie. Une terminologie est définie par une description, par des synonymes, des sources et une traduction. Tous ces éléments terminologiques sont des textes libres. Le type concept permet de représenter les objets d'intérêt du domaine étudié. Un concept est identifié par son nom qui est unique. Un concept peut avoir une terminologie, un ou plusieurs surtypes (multi-héritage), des propriétés et des axiomes. Un attribut est la réification d'une fonction. Les attributs peuvent être vus comme des concepts sans structure et avec une propriété prenant une valeur. Une expression permet d'exprimer des connaissances de type  $\langle \text{opérande} \rangle \langle \text{opérateur} \rangle \langle \text{valeur} \rangle$  où opérande est un attribut ou une propriété d'un concept. Une structure permet de définir des objets avec une structure interne. Les relations permettent d'associer les types d'objets entre eux (concept, attribut, expression, structure, et relation).

*Correspondance entre ontologies.* La notion de correspondance entre ontologies (ontology mapping) permet d'une manière informelle de définir des correspondances entre les types de deux ontologies.

*Modèle de domaine.* Un modèle de domaine est un ensemble d'expressions impliquant

les objets d'une ontologie et qui représente une perspective sur l'ontologie

**Connaissance sur le raisonnement.** La connaissance sur le raisonnement est définie par un ensemble de description d'inférences. Une inférence est décrite par le type d'opération, les rôles impliqués (intrans, extrant, statique) et par une spécification qui décrit en texte libre l'inférence.

**Connaissance sur les tâches.** La connaissance sur les tâches est définie par un ensemble de descriptions de tâches. Une tâche est spécifiée par une définition de tâche et un corps de tâche (task body). La définition de tâche précise ce que la tâche réalise. Elle est composée d'un but, des rôles impliqués dans la tâche et éventuellement d'une spécification qui décrit les dépendances logiques entre les rôles. Le corps de la tâche précise comment la tâche est réalisée. On distingue trois types de tâches : (i) les tâches composites qui sont décomposées en sous-tâches, (ii) les tâches primitives qui sont directement reliées aux inférences et (iii) les tâches de transfert qui interagissent avec les utilisateurs. Le corps de la tâche est décrit par les éléments suivants : le type de la tâche, la décomposition, la méthode de résolution (problem solving method) pour exécuter la décomposition, des rôles additionnels, les structures de contrôle et les hypothèses.

**Méthode de résolution de problèmes (Problem Solving Methods).** Pour la plupart des applications, il suffit de définir les connaissances au sujet du domaine, des inférences et des tâches. Cependant dans certaines situations, il est nécessaire de pouvoir contrôler dynamiquement l'exécution de certaines tâches. Les méthodes de résolution de problèmes permettent de le faire. Il s'agit de méta-tâches qui vont contrôler l'ordre d'exécution des sous-tâches d'une tâche. La description d'une méthode de résolution de problème s'apparente à celle d'une tâche, et on y retrouve à peu près les mêmes éléments.

La méthode CommonKADS a été développée pour la construction de systèmes experts. Et bien qu'elle se veuille une méthodologie complète de développement de systèmes à base de connaissances, elle est fortement teintée "système expert" dans son approche. Des changements certainement importants devraient être apportés pour la gestion des connaissances de l'entreprise. Cependant la partie traitant de la modélisation de domaines et de la modélisation des tâches pourraient s'appliquer assez

facilement.

## Confrontation

Le langage ou notation CML est selon les auteurs [65] un langage très structuré et semi-formel pour la spécification des modèles d'expertise de CommonKADS. Le langage que nous avons étudié couvre la connaissance de domaine (Domain Knowledge), la connaissance sur les raisonnements (Inference Knowledge) et la connaissance sur les tâches (Task Knowledge).

La confrontation consistait à savoir si le langage proposé avait la même puissance d'expression que CML. Deux possibilités s'offraient à nous pour comparer ces deux langages. La première est d'identifier un certain nombre de modèles caractéristiques de CommonKADS et de les modéliser avec notre langage. La deuxième possibilité, que nous avons choisie, est celle de la métamodélisation. La métamodélisation consiste à modéliser les composantes de modélisation de CML. Si nous pouvons représenter les composantes de modélisation, nous pouvons donc modéliser les instances de ses composantes.

Notre étude a été exhaustive. Nous avons modélisé toutes les composantes de modélisation identifiées dans [65] et nous les présentons ci-dessous.

**Hiérarchie de types** La figure 5.31 présente la hiérarchie des types de composantes des modèles de CommonKADS.

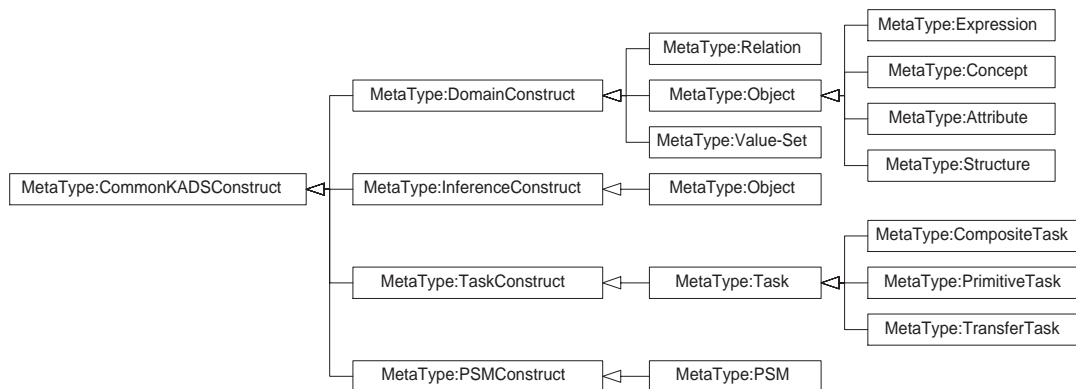


FIG. 5.31: Hiérarchie des types de CommonKADS.

**Ontologie (Ontology).** Une ontologie est définie par la spécification de types ou "construits". Ces construits peuvent être des concepts, des attributs, des expressions, des structures ou des relations. La figure 5.32 présente le graphe de spécification de *Ontology*. Afin de simplifier la représentation des graphes de spécification, nous avons utilisé la notation des fonctions introduite dans [50]. Le symbole  $\Rightarrow$  représente une fonction, c'est-à-dire une relation de cardinalité 1..1. Cette notation est un raccourci qui évite définir un graphe de restriction pour contraindre la cardinalité maximale. Dans l'exemple ci-dessous, le symbole  $\Rightarrow$  signifie qu'une entité a une et une seule description

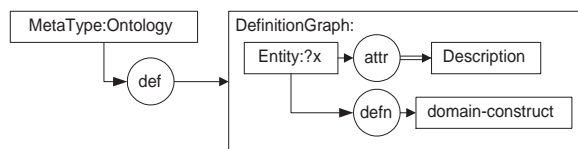


FIG. 5.32: Graphe de spécification de *Ontology*.

**Concept (Concept).** La notion de concept de CommonKADS correspond à la notion de "entité" dans le formalisme entité-relation et la notion de "classe" dans le formalisme orienté objet.

Chaque concept a un nom qui identifie le concept, un ou plusieurs super types (l'héritage multiple est autorisé par CommonKADS) et des propriétés. Une propriété est une fonction prenant ses valeurs dans un ensemble prédéfini de valeurs. La figure 5.33 présente le graphe de spécification de *Concept*.

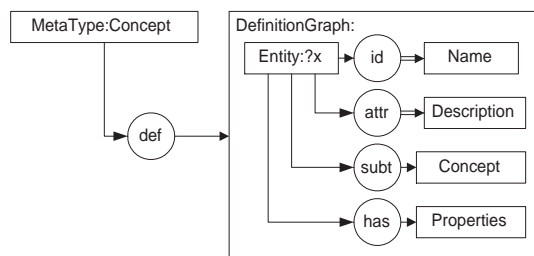


FIG. 5.33: Graphe de spécification de *Concept*.

**Attribut (Attribute).** La notion d'attribut est différente de celle des formalismes entité-relation et orienté objet. Un attribut peut être vu comme un concept sans structure interne et ayant une unique valeur.

Chaque attribut a un ou plusieurs super types. La figure 5.34 présente le graphe de spécification de Attribute.

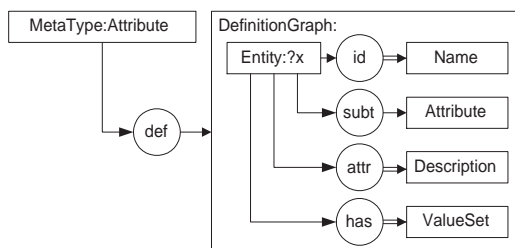


FIG. 5.34: Graphe de spécification de Attribute.

**Expression (Expression).** La notion d'expression a été introduite explicitement dans CommonKads comme un élément du langage CML, car les expressions sont très souvent utilisées dans les règles et les axiomes de domaine.

Les expressions permettent de modéliser simplement des expressions telles que  $age(patient) > 65$  ou  $pression(patient) = haute$ .

La forme générale d'une expression est  $\langle operand \rangle \langle operator \rangle \langle valeur \rangle$ . Chaque expression a un ou plusieurs super types. La figure 5.35 présente le graphe de spécification de Expression.

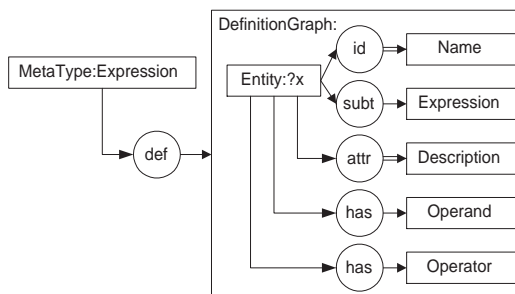


FIG. 5.35: Graphe de spécification de Expression.

**Structure (Structure).** La notion de structure est utilisée dans CML pour décrire des objets ayant une structure interne mais qui n'est pas détaillée dans un premier temps. La structure interne est représentée par une "forme" qui est un texte décrivant sommairement la structure. La figure 5.36 présente le graphe de spécification de Structure.

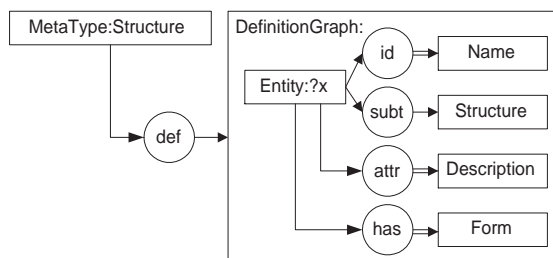


FIG. 5.36: Graphe de spécification de Structure.

**Relation (Relation).** La notion de relation est utilisée dans CML comme dans les autres formalismes de représentation. Une relation est orientée de l'argument 1 vers l'argument 2. Le nom de la relation correspond à cette orientation. La relation a également un nom inverse qui permet de lire la relation dans l'autre sens. De plus une relation peut avoir des axiomes permettant d'exprimer des règles liées à la relation. La figure 5.37 présente le graphe de spécification de Relation.

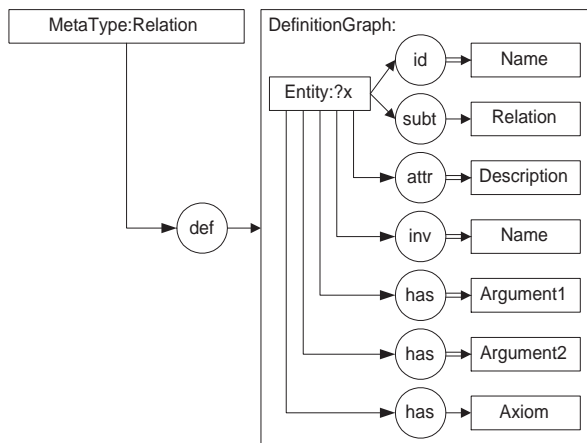


FIG. 5.37: Graphe de spécification de Relation.



**Inférence (Inference).** La notion d'inférence permet de décrire le raisonnement effectué lors de la résolution d'un problème. Le nom d'une inférence présente le rôle que cette inférence joue dans la résolution du problème. De plus, une inférence est associée à un type d'opération qui classe l'inférence. Pour chacun des rôles (intransit et extrant), une correspondance est établie avec les objets de connaissance du domaine. La figure 5.38 présente le graphe de spécification de Inference.

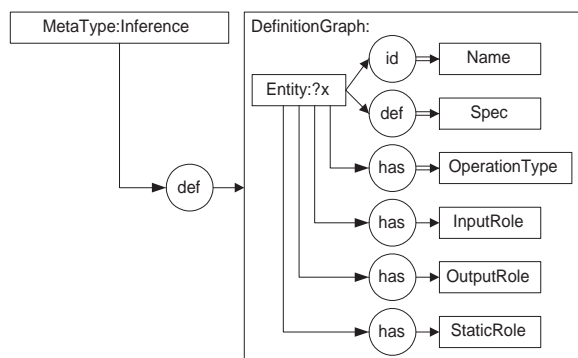


FIG. 5.38: Graphe de spécification de Inference.

**Tâche (Task).** La notion de tâche est étroitement liée à celle de but. Une tâche décrit comment le but peut être atteint. La spécification d'une tâche est composée de deux parties : la définition de la tâche et le corps de la tâche. La figure 5.39 présente le graphe de spécification de Task.

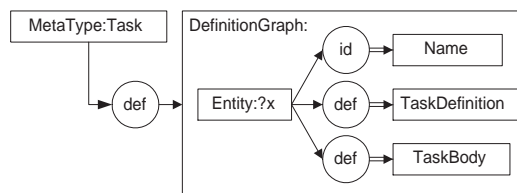


FIG. 5.39: Graphe de spécification de Task.

La définition de la tâche consiste en la déclaration du but de la tâche, des rôles impliqués dans la tâche et d'une spécification qui décrit les dépendances logiques entre les rôles de la tâche. La figure 5.40 présente le graphe de spécification de TaskDefinition.

Le corps de la tâche décrit comment le but peut être atteint. C'est une procédure décrivant les activités à exécuter pour accomplir la tâche. Le corps de la tâche est

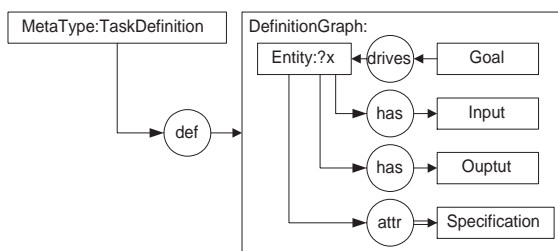


FIG. 5.40: Graphe de spécification de TaskDefinition.

défini par le type de tâche : composite ou primitive, sa décomposition si la tâche est composite, la méthode de résolution de problème qui est appliquée pour exécuter les sous-tâches, les rôles additionnels introduits par la décomposition, la structure de contrôle pour exécuter la tâche et les hypothèses liées à la décomposition.

La figure 5.41 présente le graphe de spécification de TaskBody.

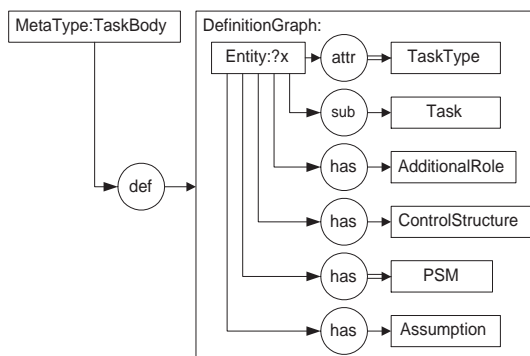


FIG. 5.41: Graphe de spécification de TaskBody.

**Méthode de résolution de problème (PSM).** La notion de méthode de résolution de problème (Problem Solving Method) permet de rendre le comportement d'une application dynamique. C'est-à-dire qu'en fonction du contexte la réalisation des tâches pourrait se faire dans un ordre différent. La figure 5.42 présente le graphe de spécification de PSM.

Une méthode de résolution de problèmes est une méta-information pour la réalisation des tâches. Une méthode sera sélectionnée pour la résolution d'une tâche lorsque le but correspondra à la compétence de la méthode et que les critères d'acceptation se-

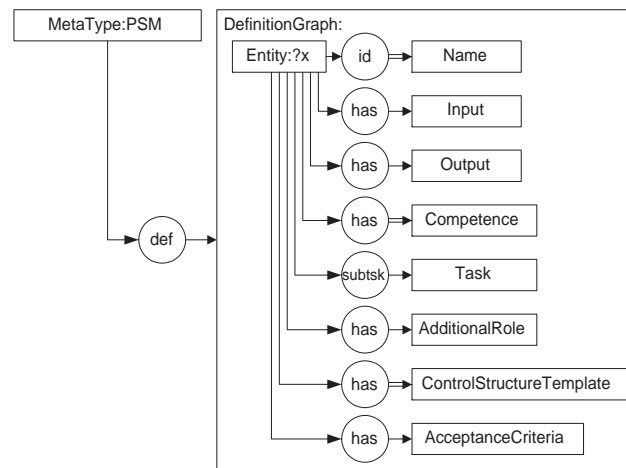


FIG. 5.42: Graphe de spécification de PSM.

ront remplis. Quand elle est appliquée, la méthode décompose la tâche en sous-tâches et introduit les rôles additionnels et utilise le gabarit de contrôle.

### 5.1.6 Évaluation théorique : Sommaire

Nous venons de voir que le modèle proposé dans la thèse répond aux besoins spécifiques identifiés pour modéliser et métamodéliser une mémoire d'entreprise. Nous avons vu également que le modèle permet d'exprimer les construits définis dans la méthode CommonKADS, méthode développée pour supporter la réalisation de systèmes experts. Sur un plan théorique, notre modèle permet donc de modéliser et de métamodéliser tout système à base de connaissances, incluant les mémoires d'entreprise.

## 5.2 Évaluation pratique

L'évaluation pratique a consisté à développer une mémoire d'entreprise à partir du modèle uniforme que nous avons défini. Cette évaluation a été réalisée dans le cadre du Groupe Conseil DMR Inc., pour lequel nous avons travaillé, et au sein duquel nous avons débuté ce travail de recherche. Le Groupe Conseil DMR Inc. avait pris conscience de l'importance stratégique que représentaient l'expérience et les connaissances acquises par ses consultants dans le domaine des technologies de l'information. Afin de tirer tous les bénéfices de cette expertise, le Groupe Conseil DMR Inc. avait

mis sur pied le projet de recherche Le Macroscopie Informatique [25] dont l'objectif était la gestion de cette expertise et des produits en découlant. Dans le cadre de ce projet nous avons développé le modèle des connaissances permettant d'acquérir, de stocker et de diffuser l'expertise entourant les méthodes développées par DMR.

Pour ce faire nous avons développé les ontologies relatives aux méthodes : méthode pour développer une stratégie des technologies de l'information, méthode pour développer une architecture technologique pour supporter la stratégie, méthode de développement de systèmes d'information pour implanter la stratégie, et enfin méthode pour gérer les bénéfices attendus de tout cela.

Les ontologies définissent le vocabulaire permettant de structurer la connaissance de l'entreprise. Le but de ce chapitre n'est pas de présenter toutes les ontologies développées dans ce projet mais de montrer que le modèle a répondu aux besoins. C'est pourquoi nous ne présentons dans ce chapitre que deux ontologies ; l'ontologie Noyau et l'ontologie Système. L'ontologie Noyau définit les notions les plus abstraites qui permettent de catégoriser l'ensemble des connaissances de l'entreprise. L'ontologie Système basée sur une vision systémique de l'entreprise définit le vocabulaire permettant de catégoriser les connaissances corporatives relatives aux processus d'affaires de l'entreprise. Cette dernière ontologie est fondamentale dans la représentation des connaissances corporatives.

### 5.2.1 Ontologie : Noyau

Cette section introduit les éléments structurant la connaissance au plus haut niveau. Cette ontologie est une adaptation et libre traduction de [69].

Nous distinguons dans l'ontologie Noyau le premier niveau qui définit les sept abstractions fondamentales sur lesquelles sont basées les autres niveaux. Lors de la définition d'une ontologie, certains types ou notions sont définis à partir des types ou notions déjà définis ; d'autres au contraire ne sont pas définis par rapport à d'autres types ou notions. Ce sont les types ou notions primitifs. Les graphes de spécification des types primitifs sont dans ce cas réduits au graphe de spécialisation.

#### Premier niveau

Le premier niveau définit trois manières orthogonales d'appréhender l'univers du discours. La première est l'opposition Physique versus Information, la deuxième est l'opposition Dynamique versus Statique, enfin la troisième distingue la notion de

Indépendant, une entité vue indépendamment des autres entités, la notion de Relatif, une entité vue en relation avec les autres entités et enfin Médiateur, une entité reliant d'autres entités. La figure 5.43 présente le premier niveau de la hiérarchie du noyau.

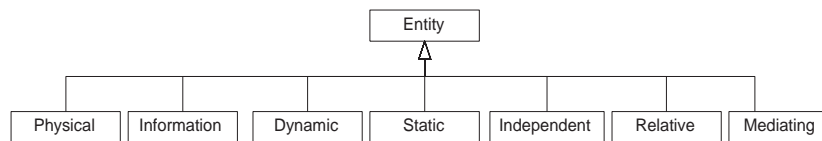


FIG. 5.43: Premier niveau de la hiérarchie de type du noyau.

Nous donnons ci-dessous pour chacun de ces types primitifs une définition textuelle et son super type.

**Entité (Entity).** Toute chose, notion ou idée. Entité est le sommet de la hiérarchie de classification et n'a donc pas de supertype. Toute chose, notion ou idée représentable est une entité.

*Supertype* : -

**Physique (Physical).** Toute entité composée de matière ou d'énergie. Une entité physique est une entité matérielle. Entité physique est définie par opposition à Information.

*Supertype* : Entity

**Information (Information).** Toute entité indépendante de toute matière et de toute énergie. Une information est une entité immatérielle, une abstraction. Les notions, les idées sont des informations.

*Supertype* : Entity

**Statique (Static).** Toute entité qui conserve son identité sur une période de temps donnée. Une entité peut être qualifiée de statique si sur une échelle de temps donnée, les attributs ou caractéristiques de l'entité sont inchangés. Cette qualification dépend de l'échelle de temps. Par exemple, une pomme sur une échelle de minutes est une entité statique. Mais sur une échelle de jours ou semaines, une pomme est une entité dynamique. Pour s'en convaincre, il suffit d'oublier une pomme sur le coin d'un bureau et de la regarder évoluer de jour en jour.

*Supertype* : Entity

**Dynamique (Dynamic).** Toute entité qui ne conserve pas son identité sur une période de temps donnée. Une entité dynamique est une entité dont les attributs ou caractéristiques évoluent sur une période de temps donnée. Dynamique est définie par opposition à Statique. Pour une même échelle de temps, il y a dichotomie entre entités statiques et entités dynamiques.

*Supertype* : Entity

**Indépendant (Independent).** Toute entité vue indépendamment des autres entités. Une entité est qualifiée d'indépendante si elle est définie indépendamment d'autres entités. Seule sa nature est considérée.

*Supertype* : Entity

**Relatif (Relative).** Toute entité définie par rapport à d'autres entités. Une entité est qualifiée de relative si sa définition est faite en rapport avec d'autres entités. Par exemple le type Femme peut être qualifié d'indépendant alors que le type Mère peut être qualifié de relatif. En effet, la notion de Mère existe par sa relation avec la notion d'Enfant.

*Supertype* : Entity

**Médiateur (Mediating).** Toute entité mettant en relation d'autres entités. Une entité est qualifiée de médiatrice si sa définition associe d'autres entités entre elles. Par exemple, Circonstance et Situation sont des entités médiatrices.

*Supertype* : Entity

### **Autres niveaux**

Les autres niveaux de l'ontologie noyau sont définis à partir des sept types du premier niveau. La figure 5.44 présente la hiérarchie sous forme d'arbre des types du noyau afin de faciliter la lecture. Seuls Physical et Information ont été représentés au premier niveau pour simplifier la hiérarchie. Le deuxième niveau présente la combinaison de physique et information avec indépendant, relatif et médiateur. Le troisième niveau est la combinaison des types du deuxième niveau avec statique et dynamique.

Les douze types du troisième niveau sont les types fondamentaux sur lesquels sont définis tous les autres types.

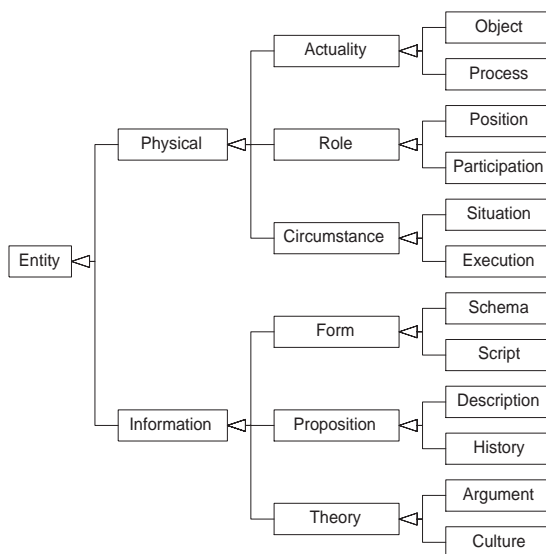


FIG. 5.44: Hiérarchie de types de l'ontologie Noyau.

Nous listons ci-dessous les définitions des types constituant l'ontologie Noyau. Les types sont présentés dans l'ordre de sous-typage illustré par la figure 5.44 (de gauche à droite et de bas en haut).

**Entité Concrète (Actuality).** Toute entité physique composée de matière ou d'énergie dont l'existence ne dépend pas d'autres entités .

*Supertype* : Physical, Independent

**Objet (Object).** Toute entité concrète et statique. Un objet conserve son identité sur une période de temps donnée.

*Supertype* : Actuality, Static

**Processus (Process).** Enchaînement ordonné d'événements aboutissant à un résultat déterminé (adapté du Petit Larousse).

*Supertype* : Actuality, Dynamic

**Rôle (Role).** Toute entité physique considérée relativement à d'autres choses.

*Supertype* : Physical, Relative

**Position (Position).** Tout rôle vu d'une manière statique.

*Supertype* : Role, Static

**Participation (Participation).** Tout rôle vu d'une manière dynamique.

*Supertype* : Role, Dynamic

**Circonstance (Circumstance).** Toute entité physique qui implique des entités physiques et des rôles.

*Supertype* : Physical, Mediation

**Situation (Situation).** Toute circonstance considérée comme statique.

*Supertype* : Physical, Mediation, Static

**Exécution (Execution).** Toute circonstance considérée comme dynamique.

*Supertype* : Physical, Mediation, Dynamic

**Form (Forme).** Toute structure d'information considérée indépendamment de toute application à quelque chose d'autre.

*Supertype* : Information, Independent

**Schéma (Schema).** Une forme qui a une structure correspondant à quelque chose de statique.

*Supertype* : Form, Static

**Script (Script).** Toute forme ayant une structure correspondant à quelque chose de dynamique.

*Supertype* : Form, Dynamic

**Proposition (Proposition).** Une proposition est une assertion sur des entités.

*Supertype* : Information, Relative



**Description (Description).** Une description est une proposition sur des aspects statiques.

*Supertype* : Proposition, Static

**Historique (History).** Un historique est une proposition concernant des aspects dynamiques.

*Supertype* : Proposition, Dynamic

**Théorie (Theory).** Une théorie est une information qui explique ou met en relation des entités.

*Supertype* : Information, Mediating

**Argument (Argument).** Un argument est une théorie au sujet d'une situation.

*Supertype* : Theory, Static

**Culture (Culture).** Une culture est une théorie au sujet d'une circonstance dynamique.

*Supertype* : Theory, Dynamic

### 5.2.2 Ontologie : Système

Cette section introduit un ensemble de concepts utilisés pour décrire et organiser la connaissance sur les situations. La terminologie a été fortement influencée par l'approche systémique présenté dans [31].

#### Types de Concept

La figure 5.45 présente la hiérarchie sous forme d'arbre de l'ontologie Système. Les concepts grisés sont repris de l'ontologie Noyau.

L'ontologie Système présentée ici est le choix retenu pour modéliser la notion de système ainsi que les entités s'y rattachant dans le but de représenter les méthodologies utilisées par les consultants. Cependant ce choix est assez générique pour modéliser des processus d'affaires plus généraux que les processus d'affaires liés à la réalisation de systèmes d'informations.

L'ontologie Système comme les autres ontologies a été modélisée grâce au modèle uniforme. Afin de ne pas trop allonger la présentation, nous présentons les graphes de

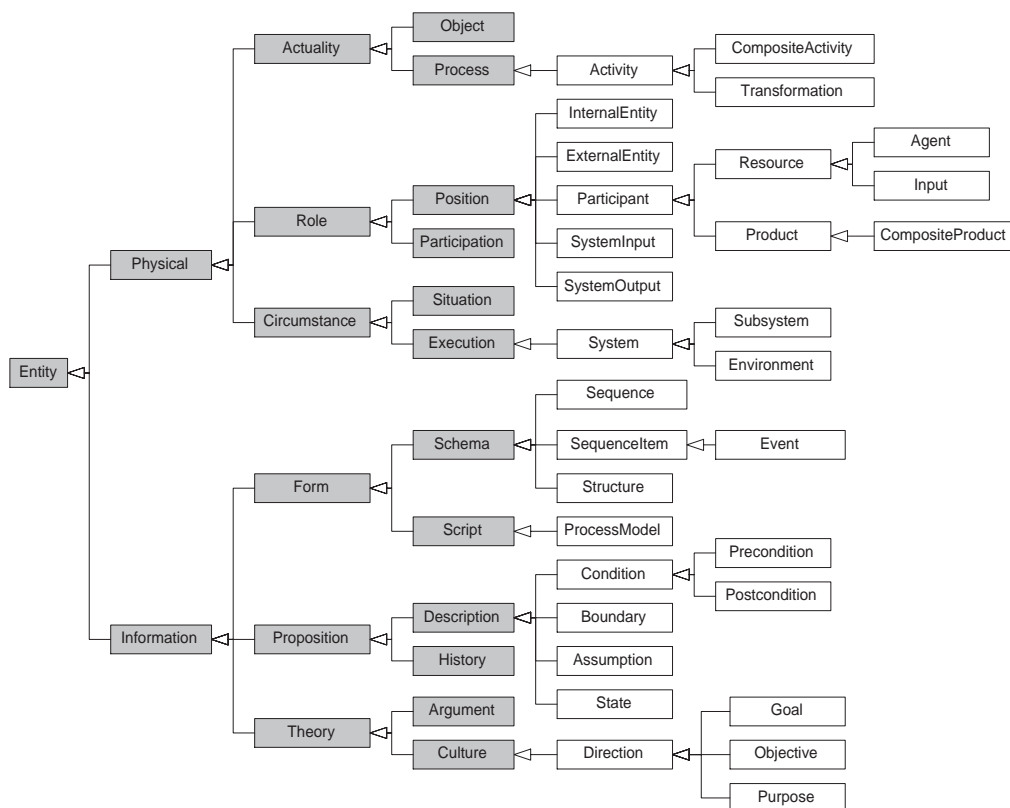


FIG. 5.45: Hiérarchie de types de l'ontologie Système.

spécifications sous forme linéaire et non sous forme graphique. L'ordre de présentation suit la hiérarchie des types de la figure 5.45.

**Activité (Activity).** Une activité est une production par des agents d'entités sortantes sous certaines conditions.

*Supertype* : Process

*Graphes de spécification* :

```
[DefinitionGraph :
  [Activity : ?x]-
    (changed) <- [Entity]
    (enables) <- [Agent]
    (ends) <- [Event]
    (depends-on) -> [Precondition]
    (realizes) -> [Postcondition] ].
```

De plus, une activité pourrait avoir les relations suivantes :

```
[Activity]-
  (used)<-[Entity]
  (governs)<-[Rule]
  (attained)<-[Objective].
```

**Activité complexe (CompositeActivity).** Une activité complexe est une activité qui est aussi définie par un modèle de processus comme une séquence d'activités.

*Supertype* : Activity

*Définition* : Graphes de spécification :

```
[DefinitionGraph :
  [Process : ?x]-
  (how)<-[ProcessModel] ].
```

```
[RestrictionGraph :
  [CompositeProcess]-
  (how)<-[ProcessModel]
  (how)<-[ProcessModel] ].
```

**Transformation (Transformation).** Une transformation est la production d'entités sortantes à partir d'entités entrantes.

*Supertype* : Activity

*Graphes de spécification* :

```
[DefinitionGraph :
  [Activity : ?x]-
  (used)<-[Entity] ].
```

**Entité interne (InternalEntity).** Une entité interne est une entité qui fait partie du système.

*Supertype* : Position

*Graphes de spécification* :

```
[DefinitionGraph :
  [Position : ?x]-
  (part)->[System] ].
```

**Entité externe (ExternalEntity).** Une entité externe est une entité qui est dans l'environnement du système.

*Supertype* : Position

*Graphes de spécification* :

```
[DefinitionGraph :
  [Position : ?x]-
  (part)->[Environment] ].
```

**Participant (Participant).** Un participant est une entité impliquée dans une activité.

*Supertype* : Position

*Graphes de spécification* :

```
[DefinitionGraph :
  [Position : ?x]-
  (involved)->[Activity] ].
```

**Ressource (Resource).** Une ressource est un participant qui est nécessaire à la réalisation d'une activité.

*Supertype* : Participant

**Agent (Agent).** Un agent est une ressource qui permet la réalisation d'un processus (adapté de [32]).

*Supertype* : Resource

*Graphes de spécification* :

```
[DefinitionGraph :
  [Resource : ?x]-
  (enables)->[Process] ].
```

**Intrant (Input).** Un intrant est une ressource utilisée par une activité.

*Supertype* : Resource

*Graphes de spécification* :

```
[DefinitionGraph :
  [Resource : ?x]-
  (used)->[Process] ].
```

**Produit ou Extrant (Product).** Un produit est une ressource résultat d'un processus (adapté de [32]). Les types de produits sont : services, matériels, matériels transformés, connaissances ou une combinaison de ceux-ci. Un produit peut être tangible (par exemple du matériel assemblé ou transformé) ou intangible (par exemple

de l'information ou des concepts) ou une combinaison. Les produits peuvent être attendus (par exemple ceux livrés aux clients) ou inattendus ( effets de bord ou non voulus) (adapté de [32]).

*Supertype* : Participant

*Graphes de spécification* :

```
[DefinitionGraph :
  [Participant : ?x]-
    (changed)->[Process] ].

[RestrictionGraph :
  [Product]-
    (changed)->[Process]
    (changed)->[Process] ].
```

**Produit Composé (Composite Product).** Un produit composé est un produit qui est aussi défini comme un ensemble de produits.

*Supertype* : Product

*Graphes de spécification* :

```
[DefinitionGraph :
  [Product : ?x]-
    (part)->[Product] ].
```

**Input du système (System Input).** Un input du système est une entité de l'environnement qui, à un moment dans le temps, franchit la frontière du système et devient une entité du système.

*Supertype* : Position

**Output du système (System Output)** Un output du système est une entité de l'environnement qui, à un moment dans le temps, franchit la frontière du système et devient une entité de l'environnement.

*Supertype* : Position

**Système (System).** Un système est une collection d'entités qui interagissent entre elles afin d'atteindre un but.

*Supertype* : Theory, Dynamic

*Graphes de spécification* :

```
[DefinitionGraph :
  [DynamicCircumstance : ?x]-
    (part)<-[Entity]
    (motivates)<-[Purpose]
    (chrc)<-[Structure]
    (chrc)<-[Environment] ].
```

```
[RestrictionGraph :
  [System]-
    (chrc)<-[Structure]
    (chrc)<-[Structure] ].
```

```
[RestrictionGraph :
  [System]-
    (chrc)<-[Environment]
    (chrc)<-[Environment] ].
```

**Sous-système (Subsystem).** Un sous-système est un système identifiable comme une composante d'un plus grand système. Un sous-système reçoit sa raison d'être du système dont il est une composante.

*Supertype* : System

*Graphes de spécification* :

```
[DefinitionGraph : [System : ?x]->(part)->[System] ].
```

**Environnement (Environment).** L'environnement d'un système est l'ensemble des entités qui sont à l'extérieur de la frontière du système et qui interagissent avec des entités du système.

*Supertype* : System

*Graphes de spécification* :

```
[DefinitionGraph :
  [System : ?x]-
    (chrc)->[System]
    (part)<-[Entity] ].
```

```
[RestrictionGraph :
  [Environment]-
    (chrc)->[System]
    (chrc)->[System] ].
```

```
[RestrictionGraph :
  [Entity]-
```

```
(part)->[System]->(chrc)->[Environment : ?x]
(part)->[Environment : *x] ].
```

**Séquence (Sequence).** Une séquence est un schéma qui structure une suite ordonnée d'éléments.

*Supertype* : Schema

*Graphe de spécification* :

```
[DefinitionGraph :
  [Schema : ?x]-
  (first)<-[SequenceItem] ].
```

**Élément de séquence (SequenceItem).** Un élément de séquence est un schéma qui réfère à une entité qu'il place dans une séquence.

*Supertype* : Schema

*Graphe de spécification* :

```
[DefinitionGraph :
  [Schema : ?x]-
  (part)->[Sequence]
  (refers)->[Entity] ].
```

**Événement (Event).** Un événement est un élément de séquence qui marque la fin d'une activité.

*Supertype* : SequenceItem

*Graphes de spécification* :

```
[DefinitionGraph :
  [SequenceItem : ?x]-
  (ends)->[Process] ].

[RestrictionGraph :
  [Event]-
  (ends)->[Process]
  (ends)->[Process] ].
```

Dans la séquence d'événements qui définit un modèle de processus, nous distinguons le ou les premiers événements des événements suivants par les relations qu'ils entretiennent.

**Premier Événement (FirstEvent).** Un premier événement est un événement qui vient en premier dans une séquence d'événements.

*Supertype* : Event

*Graphes de spécification* :

```
[DefinitionGraph :
  [Event : ?x]-
    (first)->[ProcessModel] ].

[RestrictionGraph :
  [FirstEvent]-
    (first)->[ProcessModel]
    (first)->[ProcessModel] ].
```

**Événement suivant (FollowingEvent).** Un événement suivant est un événement qui suit un ou des événements dans une séquence d'événements.

*Supertype* : Event

*Graphes de spécification* :

```
[DefinitionGraph :
  [Event : ?x]-
    (follows)->[Event] ].
```

**Structure (Structure).** La structure d'un système est la manière dont les entités du système sont en relation .

*Supertype* : Schema

**Modèle de processus (ProcessModel).** Un modèle de processus est un réseau d'événements. Le réseau d'événements décrit comment une activité complexe se réalise.

*Supertype* : Script

*Graphes de spécification* :

```
[DefinitionGraph :
  [Script : ?x]-
    (first)<-[Event]
    (how)->[Process] ].

[RestrictionGraph :
  [ProcessModel]-
    (how)->[Process]
    (how)->[Process] ].
```



**Condition (Condition).** Une condition est une proposition qui doit être vraie.

*Supertype* : Description

**Précondition (Precondition).** Une précondition est une condition sur les participants qui doit être vraie avant le début du processus.

*Supertype* : Condition

*Graphes de spécification* :

```
[DefinitionGraph :
  [Condition :?x]-
  (depends-on)<-[Process] ].
```

**Postcondition (Postcondition).** Une postcondition est une condition sur les participants qui doit être vraie à la fin de l'exécution du processus.

*Supertype* : Condition

*Graphes de spécification* :

```
[DefinitionGraph :
  [Condition :?x]-
  (realizes)<-[Process] ].
```

**Frontière (Boundary).** Une frontière est définie par une règle qui permet de distinguer parmi les entités celle qui font partie du système de celles qui n'en font pas partie .

*Supertype* : Description

**Hypothèse (Assumption).** Une hypothèse est un fait ou un prédicat considéré comme vrai.

*Supertype* : Proposition

**État (State).** Un état est une proposition qui décrit des conditions particulières ou une étape dans le cycle de vie d'un participant.

*Supertype* : Proposition

*Graphes de spécification* :

```
[DefinitionGraph :
  [Proposition :?x]-
  (attr)->[Participant] ].
```

```
[RestrictionGraph :
  [Proposition]-
    (attr)->[Participant]
    (attr)->[Participant] ].
```

**Direction (Direction).** Une direction guide, gouverne ou motive une fin (adapté de [1]) .

*Supertype* : Culture

**But (Goal).** Un but est une fin vers laquelle les activités sont dirigées. Il s'exprime en termes d'activités à réaliser (adapté de [1]) .

*Supertype* : Direction

**Objectif (Objective).** Un objectif est une étape atteignable et vérifiable dans l'atteinte d'un but par une activité (adapté de [1]) .

*Supertype* : Direction

*Graphes de spécification* :

```
[DefinitionGraph :
  [Direction :?x]-
    (attained)->[Process] ].
```

**Raison d'être (Purpose).** Une raison d'être est quelque chose établie comme une fin à atteindre. C'est la raison pour laquelle quelque chose est créée (adapté de [1]) .

*Supertype* : Direction

## Types de relations

La figure 5.46 présente la hiérarchie des types de relation sous forme d'arbre de l'ontologie Système.

Nous présentons ci-dessous les définitions des types de relation de l'ontologie Système. Pour chaque relation nous donnons sa définition, son supertype, son graphe de définition et le graphe de syntaxe dans le cas où la syntaxe de la relation diffère de sa définition. Les relations sont présentées dans l'ordre alphabétique des noms anglais des relations.

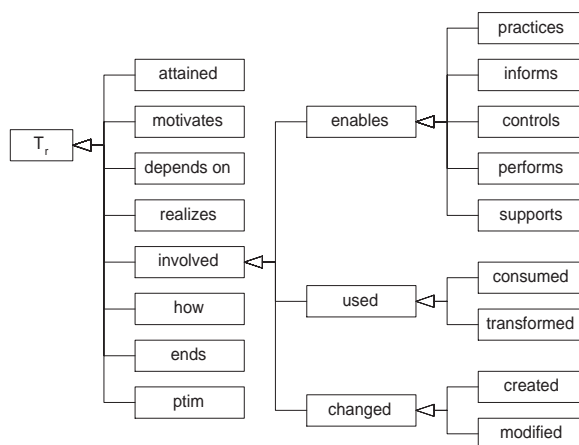


FIG. 5.46: Hiérarchie de types de relation de l'ontologie ST (System Thinking).

**est atteint (attained).** Une relation *est atteint* associe un objectif et un processus.

*Supertype* :  $T_r$

*Graphe de spécification* :

[DefinitionGraph :

[Objective : ? $x_1$ ]->(attained)->[Process : ? $x_2$ ] ].

**est changée (changed).** Une relation *est changée* associe une entité et une activité. L'entité est une transformation d'une entité utilisée par l'activité.

*Supertype* : enables

*Graphe de spécification* :

[DefinitionGraph :

[Entity : ? $x_1$ ]->(changed)->[Activity : ? $x_2$ ] ].

**est consommée (consumed).** Une relation *est consommée* associe une entité et une activité. L'entité est utilisée par l'activité et disparaît lors de l'exécution de l'activité.

*Supertype* : used

*Graphe de spécification* :

[DefinitionGraph :

[Entity : ? $x_1$ ]->(consumed)->[Activity : ? $x_2$ ] ].

**supervise (controls).** Une relation *supervise* associe une entité et une activité. L'entité guide, gère ou contrôle l'activité.

*Supertype* : enables

*Graphe de spécification* :

```
[DefinitionGraph :
  [Entity : ?x1]->(controls)->[Activity : ?x2] ].
```

**est créée (created).** Une relation *est créée* associe une entité et une activité. L'entité n'existait pas avant l'activité et prend naissance pendant l'activité.

*Supertype* : changed

*Graphe de spécification* :

```
[DefinitionGraph :
  [Entity : ?x1]->(created)->[Activity : ?x2] ].
```

**dépend de (depends-on).** Une relation *dépend de* associe une activité et une précondition. La précondition définit les circonstances dans lesquelles l'activité peut s'exécuter.

*Supertype* : T<sub>r</sub>

*Graphe de spécification* :

```
[DefinitionGraph :
  [Activity : ?x1]->(depends-on)->[Precondition : ?x2] ].
```

**permet la réalisation (enables).** Une relation *permet la réalisation* associe une entité et une activité. L'entité existe avant le début de l'activité et n'est pas modifiée par l'activité.

*Supertype* : involved

*Graphe de spécification* :

```
[DefinitionGraph :
  [Entity : ?x1]->(enables)->[Activity : ?x2] ].
```

**termine (ends).** Une relation *termine* associe un événement et une activité. L'événement marque la fin de l'activité.

*Supertype* : T<sub>r</sub>

*Graphe de spécification* :

```
[DefinitionGraph :
  [Event : ?x1]->(ends)->[Activity : ?x2] ].
```

**comment (how).** Une relation *comment* associe un modèle de processus et une activité complexe. Le modèle de processus décrit comment réaliser l'activité complexe.

*Supertype* :  $T_r$

*Graphe de spécification* :

```
[DefinitionGraph :
  [ProcessModel : ?x1]->(how)->[Activity : ?x2] ].
```

**informe (informs).** Une relation *informe* associe une entité et une activité. L'entité fournit de l'information pendant l'exécution de l'activité.

*Supertype* : enables

*Graphe de spécification* :

```
[DefinitionGraph :
  [Entity : ?x1]->(informs)->[Activity : ?x2] ].
```

**est impliquée (involved).** Une relation *est impliquée* associe une entité et une activité. L'entité est impliquée dans l'exécution de l'activité.

*Supertype* :  $T_r$

*Graphe de spécification* :

```
[DefinitionGraph :
  [Entity : ?x1]->(involved)->[Activity : ?x2] ].
```

**est modifiée (modified).** Une relation *est modifiée* associe une entité et une activité. L'entité existe avant l'exécution de l'activité et son état est modifié durant le cours de l'activité.

*Supertype* : changed

*Graphe de spécification* :

```
[DefinitionGraph :
  [Entity : ?x1]->(modified)->[Activity : ?x2] ].
```

**motive (motivates).** Une relation  *motive* associe un but et un système. Le but est la raison d'être du système.

*Supertype* :  $T_r$

*Graphe de spécification* :

```
[DefinitionGraph :
  [Purpose : ?x1]->(motivates)->[System : ?x2] ].
```

**exécute (performs).** Une relation *exécute* associe une entité et une activité. L'entité est une machine qui exécute l'activité.

*Supertype* : enables

*Graphe de spécification* :

```
[DefinitionGraph :
  [Entity : ?x1]->(performs)->[Activity : ?x2] ].
```

**pratique (practices).** Une relation *exécute* associe une entité et une activité. L'entité est un être humain qui exécute l'activité.

*Supertype* : enables

*Graphe de spécification* :

```
[DefinitionGraph :
  [Entity : ?x1]->(practices)->[Activity : ?x2] ].
```

**réalise (realizes).** Une relation *réalise* associe une activité et une postcondition. La postcondition définit les circonstances qui résultent de l'exécution de l'activité.

*Supertype* : T<sub>r</sub>

*Graphe de spécification* :

```
[DefinitionGraph :
  [Activity : ?x1]->(realizes)->[Postcondition : ?x2] ].
```

**supporte (supports).** Une relation *supporte* associe une entité et une activité. L'entité supporte la réalisation de l'activité.

*Supertype* : enables

*Graphe de spécification* :

```
[DefinitionGraph :
  [Entity : ?x1]->(supports)->[Activity : ?x2] ].
```

**est transformée (transformed).** Une relation *est transformée* associe une entité et une activité. L'entité est utilisée par l'activité et son état est modifié durant le cours de l'activité.

*Supertype* : used

*Graphe de spécification* :

```
[DefinitionGraph :
  [Entity : ?x1]->(transformed)->[Activity : ?x2] ].
```

**est utilisée (used).** Une relation *est utilisée* associe une entité et une activité. L'entité est utilisée par l'activité pendant son exécution.

*Supertype* : involved

*Graphe de spécification* :

[DefinitionGraph :

[Entity : ?x<sub>1</sub>]->(used)->[Activity : ?x<sub>2</sub>] ].

### 5.2.3 Conclusion de l'évaluation pratique

Les deux ontologies présentées ci-dessus forment la base structurant la connaissance de l'entreprise à partir de laquelle les autres ontologies : Méthode, Produit et Apprentissage, plus spécifiques au projet, ont été développées. L'ontologie Méthode définit les constituants d'une méthode (principes, activités, biens livrables, techniques, modèles, formalisme, etc.). L'ontologie Produit ne fait pas partie du domaine d'application, elle définit le vocabulaire utilisé pour structurer les connaissances méthodologiques en produits commerciaux sous forme HTML et guides PDF. Et enfin l'ontologie Apprentissage définit le vocabulaire lié à la formation. L'ensemble de ces ontologies ont permis de définir et documenter les méthodologies utilisées par les consultants du Groupe Conseil DMR dans le cadre de leurs mandats d'élaboration de plans stratégiques, d'élaboration d'architectures technologiques, de développement de systèmes et de gestion des bénéfices attendus des actions stratégiques.

Une caractéristique du projet a été le travail en parallèle de deux équipes : une équipe de développement des ontologies et du système de gestion du corpus, et une équipe de développement du corpus de connaissances. Paradoxalement, le développement du corpus de connaissances a commencé avant le développement des ontologies et du système pour accueillir ce corpus. En effet, devant l'ampleur et les multiples liens pressentis de ce corpus, il est apparu indispensable de ne pas utiliser l'approche document. Une équipe de quelques personnes a été constituée pour travailler sur une approche de type référentiel. Dans cette équipe, le rôle de l'auteur de cette thèse a été la recherche d'un formalisme de représentation de la connaissance, le développement des ontologies et un travail d'architecte dans le système de gestion. Le développement des ontologies a nécessité environ deux ans de travail avec de nombreuses interactions avec les personnes en charge du développement du corpus. Le développement du système de gestion des connaissances a été réalisé en Smalltalk et s'appuie sur le SGBD Oracle pour le stockage des connaissances.

Le cycle de production des méthodes est composé de quatre étapes :

- acquisition des ontologies,
- acquisition des connaissances,
- spécification de l'extraction et de la mise en forme des connaissances,
- extraction et mise en forme des connaissances.

La figure 5.47 montre les composantes du système [20] et illustre le cycle de production. À partir des documents structurés (à gauche) sont produits des pages HTML et des guides PDF (à droite).

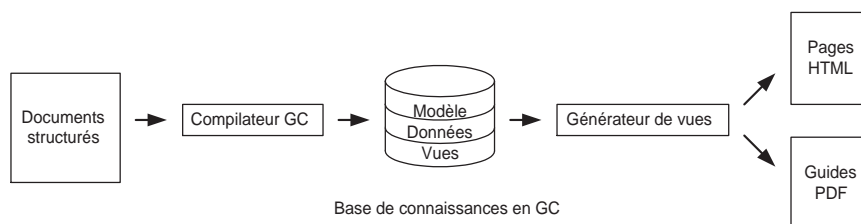


FIG. 5.47: Le cycle de production.

Les étapes d'acquisition des ontologies et des connaissances s'effectuent par des documents structurés basés sur les ontologies et non sur la présentation. Ensuite la structure des documents est analysée et le contenu est compilé en graphes conceptuels dans la base de connaissances. La troisième étape est la spécification de l'extraction et la mise en forme des connaissances. Ces spécifications, appelées vues, sont écrites aussi en graphes conceptuels et stockées dans la base de connaissances [38]. Enfin, l'extraction et la mise en forme se font par le générateur de vues qui instancie les spécifications des vues. Le système génère des pages HTML à partir des graphes conceptuels. Les guides PDF sont générés à partir d'un document SGML lui-même généré à partir des graphes. L'utilisation du même formalisme a permis de minimiser les développements et de stocker dans la même base de données les ontologies, les connaissances et les vues. Le même mécanisme permet la recherche des connaissances, la recherche des vues et leur instanciation. L'approche déclarative ainsi développée (les ontologies et les spécifications des vues ne sont pas codées "en dur") est très flexible et a permis l'évolution du métamodèle d'une manière presque continue pendant le projet.



Les méthodologies développées représentent le travail d'une équipe d'une cinquantaine de personnes pendant cinq années. À partir du corpus des connaissances modélisées en graphes conceptuels, plusieurs milliers de pages de documentation en format HTML et PDF ont été publiées. L'ensemble de cette documentation, sous format électronique constitue des produits commerciaux distribués à l'interne dans tous les bureaux du Groupe Conseil DMR (8000 consultants) et aux clients ayant acquis une licence d'utilisation des méthodes du Groupe Conseil DMR.

Quelques chiffres :

- durée du projet : 5 ans,
- nombre de personnes : environ 50 dont 5 à 7 sur le référentiel méthodologique,
- méta-métamodèle et métamodèles : 1000 types de concept,
- présentation : 2000 types de concept qui définissent comment extraire la connaissance et générer les fichiers HTML et PDF,
- méthodologies : 5 000 concepts de domaine, 20 000 biens livrables, 5 000 activités, 200 techniques,
- graphes conceptuels : 80 000 graphes,
- fichiers HTML : 100 000 fichiers français et anglais, 2000 graphiques (fichier GIF) représentant les processus et plus de 1 000 000 hyperliens.

Cependant, bien que considérable par les volumes de connaissances, il s'agit d'un cas particulier d'une mémoire d'entreprise constituée principalement de méthodes. D'autres expérimentations ont été réalisées avec les graphes conceptuels dans le domaine de l'aéronautique et dans le domaine bancaire. Dans toutes ces expérimentations, l'impact sur l'organisation n'a pas été mesuré d'une manière quantitative, mais les résultats en termes de qualité et d'organisation de la connaissance ont toujours été évalués très positivement par les clients.

### 5.3 Sommaire

Ce chapitre a présenté les diverses évaluations que nous avons faites de notre modèle. Ces évaluations ont montré qu'il répondait d'une manière théorique et d'une manière pratique aux besoins de modélisation et métamodélisation des connaissances corporatives.

L'atteinte de l'objectif 2, le modèle uniforme permet de modéliser tous les types de connaissances nécessaires à la constitution d'une mémoire d'entreprise, est atteint. De plus, les ontologies développées dans le cadre de cette évaluation sont assez générales pour modéliser et métamodéliser d'autres types de connaissances corporatives. L'approche présentée dans cette thèse est à notre connaissance la première tentative de représentation avec le même formalisme des différents niveaux de connaissances : données, modèles et métamodèles, et nous avons montré que cette approche permet de réaliser des mémoires d'entreprise de grande ampleur.

# Conclusion

Il y a un consensus sur la valeur de la connaissance corporative [63, 41]. La gestion des connaissances est devenue une préoccupation pour les grandes entreprises. Le défi est de constituer une mémoire d'entreprise où seraient entreposés toute la connaissance et le savoir-faire de l'entreprise. Aujourd'hui, les moyens de communication et de diffusion existent (réseaux et collecticiels), les outils d'acquisition textuels et graphiques sont disponibles, mais il manque les moyens adaptés à l'entreposage des connaissances corporatives. Les systèmes de gestion de données disponibles aujourd'hui gèrent efficacement les données opérationnelles, mais les natures très différentes des connaissances corporatives se prêtent mal à ces systèmes.

Nous avons vu que le problème de la gestion de la connaissance corporative était lié à la complexité et à la diversité des informations à stocker. La résolution du problème passe par la définition de plusieurs modèles exploitables à différents niveaux. Malgré l'existence de nombreux formalismes de modélisation utilisés commercialement, notre étude nous a montré qu'aucun de ces formalismes ne répondait complètement aux besoins du développement d'une mémoire d'entreprise.

Le problème le plus important est celui de l'uniformité. En effet, la construction d'un système de gestion de mémoire d'entreprise doit se faire sur un modèle uniforme, comme nous en avons vu l'illustration au chapitre 5, et non sur un ensemble de modèles avec des formalismes différents et avec tous les problèmes de traduction et de pertes d'information que cela implique dans les échanges entre modèles.

C'est pourquoi notre recherche a porté sur le développement d'un modèle uniforme permettant le développement d'un système de gestion de mémoire d'entreprise. Notre approche est basée sur le formalisme des graphes conceptuels qui par sa simplicité, sa flexibilité et sa puissance d'expression nous a permis d'atteindre le but de notre recherche, à savoir, formuler un modèle permettant la modélisation et la métamodélisation des connaissances de la mémoire d'entreprise d'une manière uniforme et intégrée.

Plus précisément, dans le cadre de cette recherche

1. Nous avons effectué un travail exhaustif de modélisation du formalisme des graphes conceptuels avec les graphes conceptuels eux-mêmes (3.2). Ceci a montré l'adéquation des graphes conceptuels pour la métamodélisation.
2. Nous avons proposé une formalisation de la notation et de la manipulation des concepts par une approche de métamodélisation, levant ainsi toute ambiguïté concernant l'utilisation des concepts (3.3).
3. Nous avons montré que le travail théorique de modélisation du formalisme des graphes conceptuels et les ontologies émergent de notre recherche constituent une base solide sur laquelle il est possible de développer un système de gestion de mémoire d'entreprise (5.1).
4. Nous avons également montré que l'utilisation du langage va au-delà de son utilisation dans la modélisation des mémoires d'entreprise et qu'il pourrait être utilisé dans d'autres domaines de modélisation d'expertise, notamment en modélisant à l'aide du langage développé dans cette thèse, les primitives de modélisation de CommonKads (5.1.5).
5. Nous avons validé le modèle uniforme proposé dans cette thèse dans plusieurs cas concrets de représentation de connaissances corporatives (5.2).

Cependant il reste encore des points importants à étudier ou résoudre.

## **Faiblesses**

Une force mais aussi une faiblesse du modèle uniforme présenté ici est la simplicité de la représentation des connaissances. C'est une force car le modèle d'implémentation n'en sera que plus facile à mettre en œuvre ; c'est une faiblesse car cette simplicité nécessite l'écriture de nombreux graphes même pour définir des types relativement simples. En particulier la représentation des cardinalités (section 5.1.1) implique l'écriture de plusieurs graphes.

Une autre faiblesse est le fait que les graphes conceptuels, bien qu'en cours de standardisation, ne sont pas aujourd'hui beaucoup utilisés dans l'industrie ; et il en existe peu de spécialistes. L'utilisation de ce formalisme dans une mémoire d'entreprise nécessite donc le développement d'interfaces conviviales aussi bien pour les ingénieurs de la connaissance chargés de développer les ontologies que pour les experts chargés d'acquérir la connaissance.

Si la thèse propose une spécification complète du langage, il reste encore des problèmes opérationnels à résoudre comme, par exemple, la classification automatique des concepts, l'extension et la contraction de concepts ou de relations. L'extension et la contraction de concepts ou de relations consiste à remplacer un concept ou une relation par le graphe de définition instancié du type du concept ou de la relation.

Il n'existe pas aujourd'hui de système commercial basé sur les graphes conceptuels. Les expériences pratiques sont donc réduites au prototype de laboratoire. Dans l'expérimentation menée chez DMR, le système n'était pas interactif et la génération des pages HTML et des documents prenait plusieurs heures. Le défi de construire un système robuste, fiable et performant est donc grand. Notre travail de recherche devra être complété par plusieurs travaux que nous présentons ci-dessous.

### **Travaux futurs**

Le travail de recherche présentée dans cette thèse n'est que la première étape conduisant à la réalisation d'une mémoire d'entreprise. Les travaux futurs porteront sur trois volets complémentaires et indispensables à cette réalisation.

Afin de situer ces travaux, nous en précisons le contexte. Nous pouvons distinguer quatre niveaux dans la mise en œuvre d'un modèle : le niveau externe (acquisition et diffusion), le niveau conceptuel, le niveau interne (implémentation) et le niveau physique (stockage). Nous avons présenté dans cette thèse le niveau conceptuel sur lequel s'appuie les autres niveaux. Le niveau externe est la représentation du niveau conceptuel pour les utilisateurs, utilisateurs chargés de l'acquisition des connaissances et utilisateurs qui consultent ou interrogent les connaissances. Les niveaux interne et physique sont la représentation informatique du niveau conceptuel.

Le premier volet porte sur les niveaux interne et physique. Il s'agit du développement, basé sur le modèle théorique présenté dans cette thèse, d'un système de gestion des connaissances performant et fiable. Le modèle d'implémentation doit être défini afin d'offrir de bonnes performances pour l'acquisition de connaissances, mais surtout pour l'interrogation et la diffusion des connaissances au travers de l'entreprise (le prototype développé chez DMR n'était pas interactif). Le système doit être fiable ; il doit assurer une validation des données.

Le deuxième volet porte sur le niveau externe et concerne l'acquisition et la diffusion des connaissances. Il s'agit de définir des interfaces conviviales pour l'ingénieur de la connaissance chargé de l'acquisition et pour les utilisateurs finaux qui vont

consulter la mémoire d'entreprise.

Enfin, le troisième volet concerne le développement d'une méthodologie pour construire et maintenir une mémoire d'entreprise, i.e., produire une méthodologie qui guidera le modélisateur dans ses activités de construction et de mise à jour d'une mémoire d'entreprise.

Pour terminer nous pensons que le modèle présenté dans cette thèse comble un besoin reconnu important quant à la représentation des connaissances de l'entreprise. Il n'existait pas à notre connaissance de modèle permettant de modéliser et métamodéliser les connaissances d'une manière uniforme. Nous avons défini un modèle

- qui répond aux besoins énoncés pour la construction d'une mémoire d'entreprise,
- qui est très général, et peut donc être utilisé dans d'autres domaines d'applications,
- qui est simple, la représentation graphique étant assez intuitive,
- qui est formel, avec un mécanisme de validation et d'inférence.

Nous croyons que ces travaux constituent un premier pas vers des techniques utiles au développement d'outils nécessaires aux entreprises pour qu'elles puissent capitaliser sur leur savoir-faire et ainsi ouvrir la voie à une nouvelle dimension dans le domaine de la gestion de données et conséquemment, des organisations.

# Références

- [1] *Webster's Ninth New Collegiate Dictionary*. Merriam-Webster Inc., 1987.
- [2] Comité JTC 1/SC 34. *ISO 8879 :1986 Traitement de l'information – Systèmes bureautiques – Langage normalisé de balisage généralisé (SGML)*. 1986.
- [3] S. Abiteboul, D. Quass, J. McHugh J., Widom, and J. Wiener. The Lorel query language for semistructured data. *International Journal of Digital Libraries*, 1(1) :68–88, 1997.
- [4] J. Barwise and J. Perry. *Situations and Attitudes*. MIT Press, Cambridge, MA, 1983.
- [5] R. Basili and G. Caldiera. Methodological and architectural issues in the experience factory. In *SEL-91-006*, pages 29–45. Software Engineering Laboratory (NASA), 1991.
- [6] G. Booch. *Object-Oriented Analysis and Design with Applications*. Benjamin/cummings, 1994.
- [7] G. Booch, J. Rumbaugh, and I. Jacobson. *Unified Modeling Language, Version 1.1*. Rational Software Corporation, 1997.
- [8] R. Brachman and J. Schmolze. An overview of the KL-One knowledge representation system. *Cognitive Science*, 9(2) :171–216, 1985.
- [9] R. J. Brachman, D. McGuinness, P. Patel-Schneider, L. Resnick, and A. Borgida. *LIVING WITH CLASSIC : When and How to Use a KL-ONE-Like Language*. Morgan Kaufmann, 1991.
- [10] P. Buneman, S. Davison, G. Hillebrand, and D. Suciu. A query language and optimization techniques for unstructured data. In H. V. Jagadish and Inderpal Singh Mumick, editors, *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 505–516, Montreal, Quebec, Canada, June 1996. ACM Press.

- [11] J. Bézivin. Who is afraid of ontologies ? In *OOPSLA '98 Workshops*, Vancouver, Canada, October 1998. available at <http://www.metamodel.com/oopsla98-cdif-workshop/bezivin1/>.
- [12] J. Bézivin, Y. Lennon, and C. Nguyen Huu Nhon. From cobol to omt : A reengineering workbench based on semantic networks. In *Proceedings of Tools USA '95*, pages 137–152, Santa Barbara, CA, USA, August 1995. Prentice Hall.
- [13] M. Chein and M-L. Mugnier. Conceptual graphs : Fundamental notions. *Revue d'Intelligence Artificielle*, 6(4) :365–406, 1992.
- [14] P. Chen. The entity-relationship model - toward a unified view of data. *ACM Transactions on Database Systems*, 1(1) :9–36, 1976.
- [15] S. Cluet, C. Delobel, J. Siméon, and K. Smaga. Your mediators need data conversion. In L. Haas and A. Tiwary, editors, *Proceedings of the ACM SIGMOD*, pages 177–188, Seattle, Washington, USA, June 1998. ACM Press.
- [16] E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6) :377–387, 1970.
- [17] E. F. Codd. Relational completeness of data base sublanguages. In R. Rustin, editor, *Data Base Systems*, number 6, pages 65–98. Prentice Hall and IBM Research Report RJ 987, San Jose, California, 1972.
- [18] CDIF Technical Committee. *CDIF -Integrated Meta-model / Foundation Subject Area - EIA/IS-111*, 1994.
- [19] NCITS.T2 Committee. *Conceptual Graph Standard - draft proposed American National Standard - NCITS.T2/98-003*, August 1999.
- [20] F. Desruisseaux, O. Gerbé, B. Guay, P. Jarest, N. Leduc, M. Perron, and P. Toussignant. CG knowledge base. In P. Eklund, G. Ellis, and G. Mann, editors, *Workshop Proceedings of the 4th International Conference on Conceptual Structures (ICCS'96)*, pages 17–18, Sydney, Australia, August 1996. University of New South Wales.
- [21] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. *XML-QL : A Query Language for XML*, 1998. <http://www.w3.org/TR/NOTE-xml-ql/>.
- [22] J. Dick. Using contexts to represent text. In W. Tepfenhart, J. Dick, and J. Sowa, editors, *Proceedings of the Second International Conference on Conceptual Structures (ICCS'94)*, pages 196–213, College Park, Maryland, USA, August 1994. Springer-Verlag.



- [23] R. Dieng, A. Giboin, C. Amergé, O. Corby, S. Després, L. Alpay, S. Labidi, and S. Lapalut. *AI magazine*, 19(4) :80–10, 1998.
- [24] DMR Consulting Group Inc. *Macroscopic Architecture : Architecture of DMR Repository*, 1994. Internal Report.
- [25] DMR Consulting Group Inc. *DMR Macroscopic*, 1996. Internal Report.
- [26] R. Elmasri and S. Navathe. *Fundamentals of Database Systems*. The Benjamin/Cummings Publishing Company Inc., 1989.
- [27] G. Engels, M. Gogolla, U. Hohenstein, and H. Conceptual modelling of database applications using an extended ER model. *Data & Knowledge Engineering*, 9(2) :157–204, 1992.
- [28] J. Esch. Contexts as white box concepts. In G. Mineau, B. Moulin, and J. Sowa, editors, *Proceedings of the 1st International Conference on Conceptual Structures (ICCS'93)*, pages 17–29, Quebec City, Quebec, Canada, August 1993. Springer-Verlag.
- [29] J. Esch. Contexts and concepts, abstraction duals. In W. Tepfenhart, J. Dick, and J. Sowa, editors, *Proceedings of the Second International Conference on Conceptual Structures (ICCS'94)*, pages 175–184, College Park, Maryland, USA, August 1994. Springer-Verlag.
- [30] J. Esch. Contexts, canons and coreferent types. In W. Tepfenhart, J. Dick, and J. Sowa, editors, *Proceedings of the Second International Conference on Conceptual Structures (ICCS'94)*, pages 185–195, College Park, Maryland, USA, August 1994. Springer-Verlag.
- [31] Robert L. Flood and Michael C. Jackson. *Creative Problem Solving : Total Systems Intervention*. John Wiley & Sons, 1991.
- [32] ISO (International Organization for Standardization). *International Standard ISO 8402, Quality Management and Quality Assurance - Vocabulary*, 1994.
- [33] G. Gardarin. *Bases de données - objet et relationnel*. Eyrolles, 1999.
- [34] O. Gerbé and B. Kerhervé. Modeling and metamodeling requirements for knowledge management. In *OOPSLA'98 Workshop*, Vancouver, Canada, October 1998. available at <http://www.metamodel.com/oopsla98-cdif-workshop/>.
- [35] O. Gerbé. Conceptual graphs for corporate knowledge repositories. In D. Lukose, H. Delugach, Keeler M., L. Searle, and J. Sowa, editors, *Proceedings of 5th*

- International Conference on Conceptual Structures (ICCS'97)*, pages 474–488, Seattle, Washington, USA, August 1997. Springer-Verlag.
- [36] O. Gerbé, B. Guay, and M. Perron. Using conceptual graphs for methods modeling. In P. Eklund, G. Ellis, and G. Mann, editors, *Auxiliary Proceedings of the 4th International Conference on Conceptual Structures (ICCS'96)*, pages 161–174, Sydney, Australia, August 1996. University of New South Wales.
- [37] O. Gerbé, R. Keller, and G. Mineau. Conceptual graphs for representing business processes in corporate memories. In *Proceedings of the sixth international Conference on Conceptual Structures (ICCS'98)*, pages 401–415. Springer-Verlag, 1998.
- [38] O. Gerbé and M. Perron. Presentation definition language using conceptual graphs. In R. Levison, editor, *Proceedings of the Peirce Workshop*, Santa Cruz, CA, USA, August 1995. University of California.
- [39] M. Goossens, F. Mittelbach, and A. Samarin. *The Latex Companion*. Addison-Wesley Pub Co, 1994.
- [40] Object Management Group. *Meta Object Facility (MOF) Specification*, 1997.
- [41] C. Havens. Enter, the chief knowledge officer. *CIO Canada*, 4(10) :36–42, 1996.
- [42] D. Hollingsworth. *The Workflow Reference Model*. Workflow Management Coalition, 1994.
- [43] Liebovitz. J., editor. CRC Press, 1999.
- [44] I. Jacobson. *Object-Oriented Software Engineering : A Use Case Driven Approach*. Addison-Wesley, 1994.
- [45] J. Le Maitre, E. Murisasco, and M. Robert. SgmlQL, un langage d'interrogation de documents SGML. In INRIA, editor, *Actes des 11èmes Journées Bases de Données Avancées (BDA '95)*, pages 431–446, Nancy, France, August 1995.
- [46] J. Lee, M. Gruninger, Y. Jin, T. Malone, A. Tate, Yost G., and other members of the PIF Working Group. *The PIF Process Interchange Format and Framework*, May 1996. <http://soa.cba.hawaii.edu/pif/>.
- [47] H. Lesca. Veille stratégique : capitalisation des signaux faibles et développement d'une intelligence collective dans l'entreprise. In *Actes de la journée d'étude ADBS*, pages 11–20, Grenoble, France, Juin 1997. Disponible à <http://www.adbs.fr/adbs/prodserv/jetude/html/prog2306.htm>.

- [48] D. Lukose. Complex modelling constructs in MODEL-ECS. In D. Lukose, H. Delugach, Keeler M., L. Searle, and J. Sowa, editors, *Proceedings of Fifth International Conference on Conceptual Structures (ICCS'97)*, pages 228–243, Seattle, Washington, USA, August 1997. Springer-Verlag.
- [49] D. Lukose and G. Mineau. A comparative study of dynamic conceptual graphs. In *Proceedings of the 11th Knowledge Acquisition Workshop (KAW98)*, volume 1, Banff, Alberta, Canada, April 1998.
- [50] G. Mineau. Mappings and functions : Essential definitions to the conceptual graph theory. In W. Tepfenhart, J. Dick, and J. Sowa, editors, *Proceedings of the Second International Conference on Conceptual Structures (ICCS'94)*, pages 160–174, College Park, Maryland, USA, August 1994. Springer-Verlag.
- [51] G. Mineau. *Conceptual Structures and Conceptual graphs*. Marcel Dekker, Inc., New York, 1996.
- [52] G. Mineau and O. Gerbé. Contexts : A formal definition of worlds of assertions. In *Proceedings of 5th International Conference on Conceptual Structures (ICCS'97)*, pages 80–94, Seattle, Washington, USA, August 1997.
- [53] G. Mineau and R. Missaoui. *Semantic Constraints in Conceptual Graph Systems*. DMR Consulting Group Inc., Montreal, Quebec, Canada, June 1996. Internal Research Report. 39 pages.
- [54] G. Mineau and R. Missaoui. The representation of semantic constraints in conceptual graph systems. In D. Lukose, H. Delugach, M. Keeler, and Sea, editors, *Proceedings of 5th International Conference on Conceptual Structures (ICCS'97)*, pages 138–152, 1997.
- [55] B. Moulin. The representation of linguistic information in an approach used for modeling temporal knowledge in discourses. In G. Mineau, B Moulin, and J. Sowa, editors, *Proceedings of 1st International Conference on Conceptual Structures (ICCS'93)*, pages 182–204, Quebec City, Quebec, Canada, August 1993. Springer.
- [56] B. Moulin. Discourse spaces : A pragmatic interpretation of contexts. In G. Ellis, R. Levinson, W. Rich, and J. Sowa, editors, *Proceedings of Third International Conference on Conceptual Structures (ICCS'95)*, pages 89–104, Santa Cruz, CA, USA, August 1995. Springer.

- [57] B. Moulin. A pragmatic representational approach of context and reference in discourses. In G. Ellis, R. Levinson, W. Rich, and J. Sowa, editors, *Proceedings of Third International Conference on Conceptual Structures (ICCS'95)*, pages 105–114, Quebec City, Quebec, Canada, August 1995. Springer.
- [58] B. Moulin and S. Dumas. The temporal structure of a discourse and verb tense determination. In J. Tepfenhart, J. Dick, and J. Sowa, editors, *Proceedings of Fourth International Conference on Conceptual Structures (ICCS'94)*, pages 45–68, College Park, Maryland, USA, August 1994. Springer-Verlag.
- [59] B. Moulin and G. Mineau. Factoring knowledge into worlds. In H. Pfeiffer, editor, *Proceedings of the 7th Annual Workshop on Conceptual Graphs*, pages 119–128, Las Cruces, New Mexico, July 1992.
- [60] M-L Mugnier. On generalization/specialization for conceptual graphs. *Journal of Experimental and Theoretical Artificial Intelligence*, 7(3) :325–344, 1995.
- [61] M-L Mugnier and M. Chein. Représenter des connaissances et raisonner avec des graphes. *Revue d'Intelligence Artificielle*, 10(1) :7–56, 1996.
- [62] C. S. Peirce. *Collected Papers of C.S. Peirce*. Harvard University Press.
- [63] C. Prahalad and G. Hamel. The core competence of the organization. *Harvard Business Review*, pages 79–91, 1990.
- [64] J. Rumbaugh, M. Blaha, W. Premerlani, and F. Eddy. *Object-Oriented Modeling and Design*. Addison-Wesley, 1991.
- [65] A. Schreiber, B. Wielenga, H. Akkermans, W. Van de Velde, and A. Anjewierden. CML : The CommonKADS conceptual modelling language. In L. Steels, A. Schreiber, and W. Van de Velde, editors, *Proceedings of the 8th European Knowledge Acquisition Workshop (EKAW'94)*, pages 1–24, Hoegaarden, Belgium, 1994. Springer-Verlag.
- [66] G. Schreiber, B. Wielenga, R. de Hoog, H. Akkermans, and W. Van de Velde. CommonKADS : A comprehensive methodology for KBS development. *IEEE Expert*, pages 28–36, December 1994.
- [67] J. Sowa. Relating diagrams to logic. In John F. Sowa Guy W. Mineau, Bernard Moulin, editor, *Proceedings of the First International Conference on Conceptual Graphs (ICCS'93)*, volume 699, pages 1–35, Quebec City, Quebec, Canada, August 1993. Springer-Verlag.

- [68] J. Sowa. Processes and participants. In P. Eklund, G. Ellis, and G. Mann, editors, *Proceedings of Fourth International Conference on Conceptual Structures (ICCS'96)*, pages 1–22, Sydney, Australia, August 1996. Springer-Verlag.
- [69] J. Sowa. *Knowledge Representation : Logical, Philosophical, and Computational Foundations*. Brooks Cole, 2000.
- [70] J. F. Sowa. *Conceptual Structures : Information Processing in Mind and Machine*. Addison-Wesley, 1984.
- [71] E. W. Stein. Organizational memory : Review of concepts and recommendations for management. *International Journal of Information Management*, 15(1) :17–32, 1995.
- [72] M. Uschold, M. King, S. Moralee, and Y. Zorgios. *The Enterprise Ontology - Technical Report AIAI-TR-195*. AIAI and University of Edinburgh, 1996.
- [73] G. van Heijst, R. van der Spek, and E. Kruizinga. Organizing corporate memories. In *Proceedings of Tenth Knowledge Acquisition for Knowledge-Based Systems (KAW96)*, Banff, Alberta, Canada, November 1996.
- [74] World Wide Web Consortium (W3C). *Extensible Markup Language (XML) 1.0*, 1998. <http://www.w3.org>.
- [75] M. Wermelinger. Conceptual graphs and first-order logic. In G. Ellis, R. Levinson, W. Rich, and J. Sowa, editors, *Proceedings of the Third International Conference on Conceptual Structures (ICCS'95)*, pages 323–337, Santa Cruz, CA, USA, August 1995. Springer-Verlag.
- [76] ISO/IEC WG11. High-level Petri nets - concepts, definitions and graphical notation - Committee draft ISO/IEC 15909. October 1997. Available at <http://www.petrinets.org/>.
- [77] B. Wielinga and A. Schreiber. Reusable and shareable knowledge bases : A european perspective. In *Proceedings International Conference on Building and Sharing of Very Large-Scaled Knowledge Bases '93*, pages 103–115, Tokyo, Japan, December 1993. Japan Information Processing Development Center.
- [78] B. Wielinga, W. Van de Velde, G. Schreiber, and H. Akkermans. The KADS knowledge modelling approach. In R. Mizoguchi, H. Motoda, J. Boose, B. Gaines, and R. Quinlan, editors, *Proceedings of the 2nd Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop*, pages 23–42, Saitama, Japan, 1992.

- [79] R. Wille. Restructuring lattice theory : an approach based on hierarchies of concepts. In I. Rival, editor, *Ordered Sets*, pages 445–470. Dordrecht-Boston : Reidel, 1982.
- [80] Workflow Management Coalition. *Interface 1 : Process Definition Interchange*, 1996. Document Number WfMC TC-1016 - Version 1.0 Beta, available at <http://www.wfmc.org>.
- [81] M. Zloff. Query-by-example : A data base language. *IBM Systems Journal*, 16(4) :324–343, 1977.

# Annexe A

## Introduction aux graphes conceptuels

Nous présentons dans cette annexe une introduction aux graphes conceptuels pour le lecteur qui ne serait pas familier avec le formalisme.

### Introduction

Les graphes conceptuels ont été introduits par John Sowa en 1984 [70]. Les graphes conceptuels allient le pouvoir d'expression de la langue naturelle avec le formalisme de la logique. Les graphes conceptuels sont un système de logique basé sur les graphes existentiels de C. S. Peirce et les réseaux sémantiques de l'intelligence artificielle. Le but du système est de représenter le sens dans une forme qui est précise, lisible et utilisable par un ordinateur. Les graphes conceptuels sont très proches de la langue naturelle comme l'illustrent les exemples ci-dessous. Les phrases *Le Canada a pour capitale Ottawa.* et *Une compagnie emploie une personne.* peuvent être modélisées par les graphes conceptuels de la figure A.1.

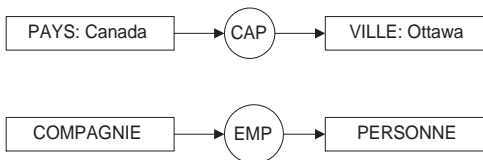


FIG. A.1: Exemple de graphes conceptuels.

Dans le premier graphe Ottawa est représentée par le concept VILLE : Ottawa , le Canada est représenté par le concept PAYS : Canada et la relation de capitale,

qui existe entre Ottawa et le Canada, est représentée par un arc joignant les deux concepts et porte le type de la relation, ici CAP pour capitale.

Dans le deuxième graphe la compagnie et la personne sont représentées respectivement par les concepts COMPAGNIE et PERSONNE et la relation d'emploi qui existe entre la personne et la compagnie est représentée par un arc dont le libellé est EMP pour emploi.

Une communauté scientifique importante travaille dans le domaine des graphes conceptuels et de nombreux travaux ont été publiés [13, 51, 49, 50, 54, 58, 60, 68, 69, 75] sur le sujet.

## Graphes conceptuels

D'une manière formelle, un graphe conceptuel est un diagramme qui représente le sens d'une phrase. Il est composé de deux sortes de nœuds, les concepts et les relations conceptuelles et d'arcs.

Les graphes conceptuels étant un système de logique, ils peuvent être traduits dans une autre forme de logique comme la logique des prédicats. Il existe en effet un opérateur  $\phi$  qui fait la traduction du formalisme des graphes conceptuels vers la logique des prédicats de premier ordre. Dans l'exemple de la figure A.1, les concepts du graphe *Une compagnie emploie une personne* sont transformés en prédicats unaires *compagnie(x)* et *personne(y)* et la relation emploie est transformée en un prédicat binaire *emploie(x, y)*. La formule logique complète est

$$(\exists x)(\exists y)((compagnie(x) \wedge personne(y) \wedge emploie(x, y))$$

Nous avons vu que dans la forme graphique, les concepts sont représentés par des rectangles, et les relations conceptuelles par des cercles et des arcs qui lient la relation aux concepts associés. Il existe une forme textuelle dite linéaire pour la représentation des graphes conceptuels. Dans la forme linéaire les concepts sont représentés entre crochets et les relations entre parenthèses, les arcs étant représentés par des flèches orientés. Les graphes de la figure A.1 s'écrivent de la manière suivante en forme linéaire

[PAYS : Canada] → (CAP) → [VILLE : Ottawa]

[COMPAGNIE] → (EMP) → [PERSONNE]



## Concepts

Les concepts représentent les objets de l'univers du discours. Ils sont représentés par des rectangles dans la forme graphique. Un concept est composé de deux parties. Il est l'assemblage d'un libellé de type et d'un référent optionnel qui identifie l'objet représenté. Suivant que l'objet référencé est connu ou inconnu, on distingue les concepts individuels et les concepts génériques.

**Concept individuel** Un concept *individuel* représente un objet déterminé (article défini ou nom propre). Il est représenté par un identifiant précédé de son type dans le rectangle. La figure A.2 présente deux exemples de concepts individuels. Le premier représente *la* ville d'Ottawa, le deuxième *l'*image #18.



FIG. A.2: Notation des concepts individuels.

**Concept générique** Un concept *générique* représente un objet indéterminé (article indéfini). Il introduit le quantificateur existentiel  $\exists$  et correspond à "Il existe un objet ...". Il est représenté par une variable précédé de son type dans le rectangle ou uniquement par le type. La figure A.3 présente deux exemples de concepts génériques. Le premier représente *une* compagnie et le deuxième *une* personne.



FIG. A.3: Notation des concepts génériques.

## Relations conceptuelles

Les relations conceptuelles permettent d'assembler les concepts pour construire une phrase, une idée, une proposition. Les relations conceptuelles sont la deuxième sorte de nœuds qui constituent un graphe conceptuel.

Sowa [] distingue trois sortes de relations conceptuelles :

- Primitive : la relation binaire LINK est la seule relation faisant partie de la théorie des graphes conceptuelles. Toutes les autres relations peuvent être construites à partir de celle-ci.
- Kit de départ : c'est l'ensemble des relations les plus couramment utilisées. On y retrouve les relations conceptuelles telles que HAS, ATTR, PART, etc.
- Définies : ce sont les relations définies pour des besoins spécifiques et qui concernent un domaine application.

Pour les relations dont les libellés sont des noms, les conventions de lecture suivantes sont souvent utilisées. Quand le graphe est lu dans le sens des flèches, l'arc dirigé vers la relation est lu "a un(e)", et l'arc qui est issu de la relation est lu "qui est". Quand le graphe est lu dans le sens contraire des flèches, l'arc issu de la relation est lu "est un(e)" et l'arc dirigé vers la relation est lu "de".

Par exemple, le graphe suivant représente la phrase *Le Canada a pour capitale Ottawa*.

[PAYS : Canada] → (CAP) → [VILLE : Ottawa].

La lecture du graphe dans le sens des flèches est *Le pays Canada a une capitale qui est Ottawa*. La lecture dans le sens contraire des flèches est *Ottawa est une capitale du pays Canada*.

Bien que la plupart des relations conceptuelles soient binaires, il est possible de définir des relations n-aires. Par exemple, la relation ENTRE est une relation tertiaire. Les deux premiers arcs sont issus des objets qui encadrent le troisième. Le graphe suivant représente la phrase *John est entre un arbre et une voiture*.

[PERSONNE : John] ← (ENTRE) -  
 1 ← [ARBRE]  
 2 ← [VOITURE].

## Référents

Les concepts représentent le sens des mots du discours. Les concepts sont des interprétations des entités, des actions, des propriétés ou des événements du monde réel. Une interprétation est constituée d'un type et d'un référent. Le type catégorise l'objet représenté. Le référent représente et identifie l'objet de l'univers du discours dont le concept est une interprétation.

Les référents des concepts `VILLE : Ottawa` et `PAYS : Canada` sont respectivement Ottawa et Canada.

## Types de concept

Les concepts peuvent être catégorisés en se basant sur le type des relations conceptuelles qu'ils entretiennent avec d'autres concepts. Comme les classes de UML définissent les catégories, les types de concept, dans le formalisme des graphes conceptuelles, définissent les catégories. Les types de concept sont organisés en une hiérarchie de type. La hiérarchie de type est un ordre partiel défini sur l'ensemble des libellés de type. la relation d'ordre partiel de généralisation définit un treillis de types avec le type  $\top$ , qui est le type universel, au sommet et le type  $\perp$ , qui est le type absurde à sa base. Tout concept est de type  $\top$  et il n'existe aucun concept de type  $\perp$ .

Un type de concept est spécifié par un graphe de définition auquel tout instance du concept doit se conformer. Le graphe suivant montre le graphe de définition du type EMPLOYÉ; ce graphe signifie qu'un employé est une personne qui travaille pour une organisation.

```
Type EMPLOYÉ(x) is
  [PERSONNE : *x] ← (EMP) ← [COMPAGNIE].
```

## Types de relation

Comme les types de concept, les types de relation sont spécifiés par un graphe de définition auquel tout instance de la relation doit se conformer.

Considérons le graphe ci-dessous qui représente la phrase *Le chat Garfield mange des lasagnes*.

```
[CHAT : GARFIELD] → (MANGE) → [LASAGNE].
```

La relation MANGE lie Garfield à la nourriture qu'il est en train de manger. Plus généralement nous définissons MANGE par le graphe suivant qui montre le graphe de définition du type de relation MANGE.

```
Type MANGE(x,y) is
  [ETRE_ANIMÉ : *x] → (AGNT) → [MANGER] ← (OBJ) ← [NOURRITURE : *y].
```

Le graphe exprime qu'une relation MANGE lie un être animé qui est l'agent d'une action MANGER qui a pour objet de la nourriture.

Les hiérarchies de types et de relations forment respectivement deux treillis. Ces treillis définissent l'héritage entre les types. Le formalisme des graphes conceptuels supporte l'héritage multiple. Un type peut être sous-type de plusieurs types.

## Contraction et expansion

Les types de concept et les types de relation sont utilisés pour valider l'acquisition de connaissances et pour les opérations de contraction et d'expansion. Ces opérations sont des opérations fondamentales dans le traitement et l'explication de la connaissance. L'opération de contraction consiste à remplacer un graphe correspondant à une définition de type de concept par un concept de ce type ou à remplacer un graphe de définition de type de relation par une relation de ce type. L'opération de contraction permet d'exprimer d'une manière plus concise la connaissance.

Soit la phrase *Une compagnie emploie Jean*, elle est représenté par le graphe

$$[\text{COMPAGNIE}] \rightarrow (\text{EMP}) \rightarrow [\text{PERSON} : \text{JEAN}].$$

Celui-ci traduit le fait que Jean est un employé. L'opération de contraction appliquée au graphe ci-dessous donne une forme plus précise et compacte de la même information.

$$[\text{EMPLOYÉ} : \text{JEAN}].$$

L'expansion est l'opération inverse et consiste à remplacer un concept ou une relation par son graphe de définition. L'opération d'expansion est l'opération qui permet de donner des explications, de détailler la connaissance. Par exemple, l'expansion du graphe

$$[\text{CHAT} : \text{GARFIELD}] \rightarrow (\text{MANGE}) \rightarrow [\text{LASAGNE}].$$

donne le graphe

$$[\text{CHAT} : \text{GARFIELD}] \rightarrow (\text{AGNT}) \rightarrow [\text{MANGER}] \leftarrow (\text{OBJ}) \leftarrow [\text{LASAGNE}].$$

## Liens de co-références

Deux concepts peuvent représenter le même objet sans avoir pour autant la même représentation. Soit la phrase *Jean est l'homme qui a vu l'ours*, Jean et l'homme représentent le même objet à savoir Jean mais leur représentation est différente. Jean est représenté par le concept  $[\text{PERSONNE} : \text{JEAN}]$  et l'homme est représenté par le

concept HOMME. Deux concepts qui représentent le même objet sont dits co-référents. Graphiquement des concepts co-référents sont joints par une ligne pointillée appelée *lien de coréférence*. La figure A.4 représente la phrase *Jean est l'homme qui a vu l'ours*, le graphe montre un lien de coréférence entre les concepts [PERSONNE : JEAN] et [HOMME].

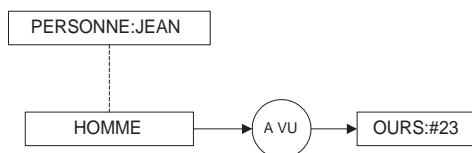


FIG. A.4: Lien de coréférence.

Dans la forme linéaire des variables sont nécessaires pour montrer le lien de coréférence. Le graphe de la figure A.4 est représenté par

$$[\text{PERSON} : \text{JEAN} ?x] [\text{HOMME} : *x] \rightarrow (\text{A VU}) \rightarrow [\text{OURS} : 23].$$

## Situation et contexte

Une situation [4] est une configuration finie de certains aspects du monde dans une région limitée du temps et de l'espace. Une situation peut être statique et rester inchangée pendant une période de temps ou inclure des processus et des événements qui vont causer des changements. Une situation peut être réelle ou imaginaire, présente, passée ou future.

Une situation est représentée dans la théorie des graphes conceptuels par un *contexte* qui est un concept qui contient une ou plusieurs propositions qui décrivent la situation. La figure A.5 illustre la situation *Jean pense que Marie veut épouser un marin*.

Le lien de coréférence entre [PERSONNE : MARIE] et [PERSONNE] montre que la personne dans la situation la plus emboîtée réfère au même individu que le concept [PERSONNE : MARIE] dans le contexte englobant. La forme linéaire est :

$$\begin{aligned} &[\text{PERSONNE} : \text{JEAN}] \rightarrow (\text{PENSE}) \rightarrow [\text{PROPOSITION} : \\ &\quad [\text{PERSONNE} : \text{MARIE} ?x] \rightarrow (\text{VEUT}) \rightarrow [\text{SITUATION} : \\ &\quad\quad [\text{PERSONNE} : *x] \rightarrow (\text{MARIER}) \rightarrow [\text{MARIN} ] ] ]. \end{aligned}$$

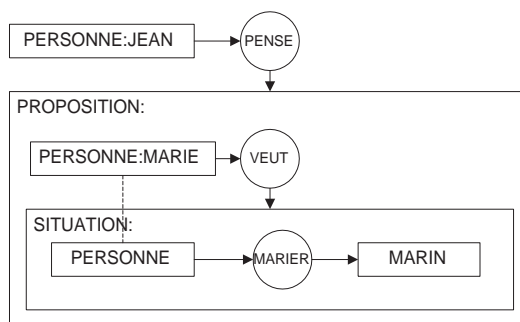


FIG. A.5: Proposition et contexte emboîtés.

Le lien de coréférence est représenté par des variables. La variable  $?x$  est associée au concept qui détermine l'individu référencé, la variable  $*x$  montre que le concept  $[PERSONNE]$  est co-référent avec le concept  $[PERSONNE : MARIE ?x]$ .

L'opérateur  $\phi$  préserve les liens de coréférences. La formule logique équivalente de la figure A.5 est

$$\begin{aligned}
 &(personne(Jean) \wedge pense(Jean, \\
 &\quad (personne(Marie) \wedge veut(Marie, \\
 &\quad\quad (\exists x)(marier(Marie, x) \wedge marin(x))))))
 \end{aligned}$$

## Conclusion

Cette annexe n'est qu'une brève introduction au formalisme des graphes conceptuels. Cette thèse approfondit et en formalise une partie, le langage lui-même. Le lecteur désireux de faire une étude plus approfondie des graphes conceptuels lira avec intérêt les deux livres de John Sowa [70, 69] qui détaillent le formalisme des graphes conceptuels et discutent de la problématique de la représentation de la connaissance dans les ordinateurs.