



HAL
open science

Adaptation d'ontologies avec les grammaires de graphes typés : évolution et fusion

Mariem Mahfoudh

► **To cite this version:**

Mariem Mahfoudh. Adaptation d'ontologies avec les grammaires de graphes typés : évolution et fusion. Autre [cs.OH]. Université de Haute Alsace - Mulhouse, 2015. Français. NNT : 2015MULH1519 . tel-01528579

HAL Id: tel-01528579

<https://theses.hal.science/tel-01528579>

Submitted on 29 May 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

École Doctorale 269 Mathématiques, Sciences de l'Information et de l'Ingénieur (MSII)
Laboratoire Modélisation Intelligence Processus et Systèmes (MIPS)

Thèse présentée pour obtenir le grade de
Docteur de l'Université de Haute Alsace
Discipline : Informatique

Adaptation d'ontologies avec les grammaires de graphes typés : évolution et fusion

Par : Mariem Mahfoudh

Soutenue publiquement le 29 mai 2015

Membres du jury

Rapporteur : Ladjel Bellatreche, Professeur, Université de Poitiers

Rapporteur : Christophe Nicolle, Professeur, Université de Bourgogne

Examineur : Amir Hajjam, Maître de Conférences HDR, Université de Belfort

Directeur de thèse : Michel Hassenforder, Professeur, Université de Haute Alsace

Examineur : Laurent Thiry, Maître de Conférences, Université de Haute Alsace

Examineur : Germain Forestier, Maître de Conférences, Université de Haute Alsace

خلقت طليقا كطيف النسيم
وحرا كنور الضحى في سماه

ألا انهض و سر في سبيل الحياة
فمن نام لم تنتظره الحياة

إلى النور فالنور عذب جميل
إلى النور فالنور ظل الإله

ابو القاسم الشابي

Résumé

Étant une représentation formelle et explicite des connaissances d'un domaine, les ontologies font régulièrement l'objet de nombreux changements et ont ainsi besoin d'être constamment adaptées pour notamment pouvoir être réutilisées et répondre aux nouveaux besoins. Leur réutilisation peut prendre différentes formes (évolution, alignement, fusion, etc.), et présente plusieurs verrous scientifiques. L'un des plus importants est la préservation de la consistance de l'ontologie lors de son changement. Afin d'y répondre, nous nous intéressons dans cette thèse à étudier les changements ontologiques et proposons un cadre formel capable de faire évoluer et de fusionner des ontologies sans affecter leur consistance.

Premièrement, nous proposons TGGOnto (Typed Graph Grammars for Ontologies), un nouveau formalisme permettant la représentation des ontologies et leurs changements par les grammaires de graphes typés. Un couplage entre ces deux formalismes est défini afin de profiter des concepts des grammaires de graphes, notamment les NAC (Negative Application Conditions), pour la préservation de la consistance de l'ontologie adaptée. Deuxièmement, nous proposons EvOGG (Evolving Ontologies with Graph Grammars), une approche d'évolution d'ontologies qui se base sur le formalisme GGTonto et traite les inconsistances d'une manière a priori. Nous nous intéressons aux ontologies OWL et nous traitons à la fois : (1) l'enrichissement d'ontologies en étudiant leur niveau structurel et (2) le peuplement d'ontologies en étudiant les changements qui affectent les individus et leurs assertions. L'approche EvOGG définit des changements ontologiques de différents types (élémentaires, composées et complexes) et assure leur implémentation par l'approche algébrique de transformation de graphes, SPO (Simple PushOut). Troisièmement, nous proposons GROM (Graph Rewriting for Ontology Merging), une approche de fusion d'ontologies capable d'éviter les redondances de données et de diminuer les conflits dans le résultat de fusion. L'approche proposée se décompose en trois étapes : (1) la recherche de similarité entre concepts en se basant sur des techniques syntaxiques, structurelles et sémantiques ; (2) la fusion d'ontologies par l'approche algébrique SPO ; (3) l'adaptation de l'ontologie globale résultante par le biais des règles de réécriture de graphes.

Afin de valider les travaux menés dans cette thèse, nous avons développé plusieurs outils open source basés sur l'outil AGG (Attributed Graph Grammar). Ces outils ont été appliqués sur un ensemble d'ontologies, essentiellement sur celles développées dans le cadre du projet européen CCAIps (Creatives Companies in Alpine Space) qui a financé les travaux de cette thèse.

Abstract

Ontologies are a formal and explicit knowledge representation. They represent a given domain by their concepts and axioms while creating a consensus between a user community. To satisfy the new requirements of the represented domain, ontologies have to be regularly updated and adapted to maintain their consistency. The adaptation may take different forms (evolution, alignment, merging, etc.), and represents several scientific challenges. One of the most important is to preserve the consistency of the ontology during the changes. To address this issue, we are interested in this thesis to study the ontology changes and we propose a formal framework that can evolve and merge ontologies without affecting their consistency.

First we propose TGGOnto (Typed Graph Grammars for Ontologies), a new formalism for the representation of ontologies and their changes using typed graph grammars (TGG). A coupling between ontologies and TGG is defined in order to take advantage of the graph grammars concepts, such as the NAC (Negative Application Conditions), in preserving the adapted ontology consistency. Second, we propose EvOGG (Evolving Ontologies with Graph Grammars), an ontology evolution approach that is based on the TGGOnto formalism that avoids inconsistencies using an a priori approach. We focus on OWL ontologies and we address both : (1) ontology enrichment by studying their structural level and (2) ontology population by studying the changes affecting individuals and their assertions. EvOGG approach defines different types of ontology changes (elementary, composite and complex) and ensures their implementation by the algebraic approach of graph transformation, SPO (Single pushout). Third, we propose GROM (Graph Rewriting for Ontology Merging), an ontologies merging approach that avoids data redundancy and reduces conflict in the merged result. The proposed approach consists of three steps : (1) the similarity search between concepts based on syntactic, structural and semantic techniques ; (2) the ontologies merging by the algebraic approach SPO ; (3) the global ontology adaptation with graph rewriting rules.

To validate our proposals, we have developed several open source tools based on AGG (Attributed Graph Grammar) tool. These tools were applied to a set of ontologies, mainly on those developed in the frame of the CCAIps (Creatives Companies in Alpine Space) European project, which funded this thesis work.

Remerciements

Je tiens tout d'abord à remercier Michel Hassenforder d'avoir dirigé cette thèse et de m'avoir accueilli au sein de son équipe GL. Je voudrais le remercier aussi pour la grande liberté d'action qu'il m'a donné et le cadre de travail qu'il m'a fourni durant mes trois ans de thèse. Je tiens à saluer ses conseils et son investissement dans le projet Européen CCAIps qui a financé mes travaux de recherche.

Je remercie également mon encadrant Laurent Thiry qui m'a fait découvrir la théorie des catégories. C'est grâce à son expertise dans ce domaine que j'ai pu mieux assimiler cette partie de thèse. Je voudrais le remercier aussi pour ses discussions enrichissantes et son œil critique qui m'a poussé à donner le meilleur de moi-même.

Je remercie aussi mon encadrant Germain Forestier pour ses discussions constructives et ses conseils pertinents. Je salue vivement son soutien, son suivi et sa disponibilité. Nos échanges m'ont ouvert les yeux sur bien des choses dans le domaine de recherche et m'ont permis d'apprendre de son expérience et de sa motivation que j'ai toujours admirées.

Mes remerciements s'adressent aussi à Olivier Haeberlé qui a accepté de m'accueillir au sein de son laboratoire MIPS.

Et parce que la thèse n'est pas seulement le résultat de trois ans de travail mais c'est aussi le fruit de tout un cursus académique et professionnel, je voudrais adresser mes sincères remerciements à tous mes enseignants de la Tunisie qui ont contribué à ma formation. Une pensée particulière aux membres du laboratoire MIRACL, à Faiez Gargouri et à Wassim Jaziri qui a guidé mes premiers pas dans la recherche et m'a fait découvrir le domaine des ontologies.

Mes remerciements s'adressent également à Ladjel Bellatreche et Christophe Nicolle pour l'honneur qu'ils m'ont fait d'avoir accepté rapporter cette thèse et pour l'intérêt qu'ils ont porté à mon travail.

Je remercie aussi Amir Hajjam d'avoir accepté faire partie de mon jury et d'évaluer mon travail.

Je tiens aussi à remercier tous les membres de l'équipe GL. Je remercie en particulier mes amis, les doctorants, Houda, Florent et Trung.

Merci à Houda avec qui j'ai partagé des moments de joie mais aussi d'inquiétude et de stress.

Merci à Florent pour l'agréable ambiance qu'il apporte les jours où il vient au labo. Une personne sympa avec qui j'aime bien discuter et échanger.

Merci à Trung qui a donné une ambiance anglophone à notre équipe.

Une pensée particulière aux chercheurs et ingénieurs, Marwa, Sébastien et Rushnadra, qui ont travaillé à notre équipe.

Merci à Marwa, ma chère puce. Depuis notre connaissance, elle n'a jamais cessé de me soutenir et de m'encourager. Merci pour tous les moments que nous avons partagés ensemble.

Merci à Sébastien, le grand voyageur "Ibn Battuta" comme j'aime bien l'appeler. J'ai toujours admiré son ouverture d'esprit et j'ai beaucoup appris de lui.

Merci à Rushandra, la personne la plus généreuse que j'ai jamais rencontrée.

Je remercie également mes amis les doctorants et les docteurs de l'équipe de MIAM avec qui j'ai partagé des agréables moments : Fatiha, Hamdi, Hosni, Rachid, Oussema et plus particulièrement Sabra qui m'a accompagnée durant ces années de thèse.

Mes remerciements tout particulier à mes très chers parents Fouzia et Nouredine. C'est grâce à votre amour, votre confiance, votre soutien, ... que j'ai pu réaliser cette thèse. Aucun mot ne saurait exprimer mon profond amour, mon respect et ma reconnaissance.

Je remercie également ma sœur Maha et mon frère Mohammed Amine. Merci mes adorables pour votre encouragement, vos plaisanteries, ..., pour votre présence dans ma vie.

Merci à mon neveu Ahmed pour l'agréable ambiance qu'il a apportée à notre famille. Son mot magique "tata" et son sourire formidable innocent sont capables de me faire oublier n'importe quel souci.

Merci à ma grande famille et à tous mes ami(e)s. Une pensée très particulière à ma chère Jihen, mon amie et ma sœur, pour son soutien et son encouragement. Un grand merci également à Nizar et Omar, mes amis et mes frères, pour leurs conversations et leur encouragement.

Enfin, merci à toutes et à tous qui ont contribué de près ou de loin pour la réalisation de cette thèse.

Table des matières

1	Introduction Générale	19
1.1	Introduction	19
1.2	Motivation et définition du problème	20
1.2.1	L'évolution d'ontologies	20
1.2.2	La fusion d'ontologies	22
1.2.3	Les grammaires de graphes	22
1.3	Approches proposées et contributions	23
1.4	Organisation du manuscrit	24
1.5	Publications	25
	Première partie :	
	État de l'art	27
2	Réutilisation d'ontologies	29
2.1	Introduction	30
2.2	Langages de représentation d'ontologies	31
2.2.1	Logiques de description (LD)	32
2.2.2	Resource Description Framework (RDF)	34
2.2.3	Resource Description Framework Schema (RDFS)	34
2.2.4	Web Ontology Language (OWL)	35
2.3	Réutilisation d'ontologies	37
2.3.1	Cycle de vie des ontologies	37
2.3.2	Enjeux de la réutilisation d'ontologies	39
2.3.3	Cohérence d'ontologies	41
2.4	Évolution d'ontologies	43
2.4.1	Besoins de l'évolution	43
2.4.2	Classification des changements ontologiques	44
2.4.3	Approches d'évolution d'ontologies	45
2.5	Fusion d'ontologies	53
2.5.1	Besoins de fusion	53

2.5.2	Stratégies de fusion	54
2.5.3	Approches de fusion d'ontologies	55
2.6	Bilan et positionnement	58
3	Grammaires de graphes	61
3.1	Introduction	61
3.2	Graphes et Grammaires de Graphes	62
3.2.1	Préliminaires	62
3.2.2	Grammaires de Graphes	64
3.3	Approches algébriques de transformations de graphes	68
3.3.1	Théorie des Catégories	68
3.3.2	Approche Double PushOut (DPO)	70
3.3.3	Approche Simple PushOut (SPO)	71
3.3.4	Exemple d'application du SPO sur les GGT	72
3.4	Outils de transformation de graphes	74
3.4.1	FUJABA	74
3.4.2	VIATRA	74
3.4.3	ATOM3	74
3.4.4	AGG	75
3.5	Applications des grammaires de graphes aux ontologies	75
3.5.1	Travaux liés à l'interrogation d'ontologies	76
3.5.2	Travaux liés à l'évolution d'ontologies	76
3.5.3	Travaux liés à la fusion d'ontologies	77
3.6	Bilan	78
Deuxième partie :		
Propositions		81
4	Approche d'évolution d'ontologies	83
4.1	Introduction	83
4.2	Approche d'évolution d'ontologies	85
4.2.1	Modèle de représentation d'ontologies et de leurs changements	86
4.2.2	Processus global d'évolution d'ontologies	88
4.3	Formalisation des changements élémentaires	93
4.3.1	Changements de renommage	95
4.3.2	Changements d'ajout	96

4.3.3	Changements de suppression	97
4.4	Formalisation des changements composés	97
4.4.1	Changements d'ajout	98
4.4.2	Changements de suppression	99
4.5	Formalisation des changements complexes	101
4.5.1	Changements de déplacement	102
4.5.2	Changements de fusion	102
4.5.3	Changements de division	103
4.6	Caractéristiques de l'approche proposée	104
4.7	Bilan	106
5	Approche de fusion d'ontologies	107
5.1	Introduction	107
5.2	Approche de fusion d'ontologies	108
5.2.1	Exemples introductifs	108
5.2.2	Processus global de fusion d'ontologies	110
5.3	Recherche de similarité	112
5.4	Fusion des ontologies	113
5.4.1	Appariement d'ontologies	114
5.4.2	Construction de l'ontologie commune	115
5.4.3	Construction de l'ontologie globale	116
5.5	Adaptation de l'ontologie globale	116
5.5.1	Enrichissement par des relations de synonymies	117
5.5.2	Enrichissement par des relations de subsumptions	117
5.6	Bilan	119
6	Application aux ontologies CCAIps	121
6.1	Introduction	121
6.2	Cadre applicatif du travail	122
6.2.1	Ontologies CCAIps	122
6.2.2	Changements ontologiques dans les ontologies CCAIps	125
6.3	Implémentation	128
6.3.1	Les outils OWLToGGX et GGXToOWL	128
6.3.2	Approche EvOGG	131
6.3.3	Approche GROM	132
6.4	Résultats et évaluation	133
6.4.1	Transformation d'ontologies OWL en graphes hôtes	133

6.4.2	Approche d'évolution	133
6.4.3	Approche de fusion	136
6.5	Bilan	139
7	Conclusion	141
7.1	Contributions	142
7.2	Perspectives	143
7.2.1	Perspectives à court terme	143
7.2.2	Perspectives à moyen terme	144
7.2.3	Perspectives à long terme	145
	Annexes	147
A	Formalisation des changements ontologiques avec les grammaires de graphes	149
B	Journalisation	155

Table des figures

2.1	Exemple d'une ontologie.	31
2.2	Exemple d'une ontologie représentée avec le langage AL.	33
2.3	Famille des logiques de description [Gagnon, 2007].	33
2.4	Exemple d'un graphe RDF	34
2.5	Exemple d'une ontologie OWL représentée avec la syntaxe RDF/XML.	37
2.6	Cycle de vie d'une ontologie.	38
2.7	Exemples d'inconsistances et d'incohérences.	41
2.8	L'impact des changements ontologiques sur les entités de l'ontologie.	44
3.1	Exemples des types de graphes.	63
3.2	Morphisme de graphes.	64
3.3	Principe de transformation de graphes.	65
3.4	Transformation de graphes selon l'approche "Node Label Controlled".	66
3.5	Transformation de graphes selon l'approche "Edge Replacement".	67
3.6	Les concepts span et cospan de la théorie des catégories.	69
3.7	Les concepts pushout et pullback de la théorie des catégories.	69
3.8	Les étapes de l'application de l'approche DPO.	70
3.9	Application d'une règle de réécriture selon L'approche SPO.	72
3.10	Relation de typage avec le jeu de dame.	73
3.11	Exemple d'application d'une règle de réécriture avec l'approche SPO.	73
4.1	Approche proposée pour l'évolution d'ontologies.	84
4.2	Relation entre ontologies et grammaires de graphes typés.	85
4.3	Graphe type utilisé pour décrire les ontologies.	86
4.4	Exemple d'ontologie conforme au graphe type de la Figure 4.3.	87
4.5	Processus global de l'approche d'évolution d'ontologies EvOGG.	88
4.6	Taxonomie des changements étudiés.	90
4.7	Représentation et application de la règle de réécriture du changement <i>AddIndividual</i> avec l'approche SPO.	93
4.8	Règle de réécriture du changement <i>RenameIndividual</i>	95

4.9	Règle de réécriture du changement <i>AddDisjointClasses</i>	96
4.10	Règle de réécriture du changement <i>RemoveEquivalentObjectProperties</i>	97
4.11	Règle de réécriture du changement <i>AddClass</i> avec l'alternative de correction <i>AddEquivalentClasses</i>	99
4.12	Règle de réécriture du changement <i>RemoveObjectProperty</i>	100
4.13	Changements dérivés <i>RemoveIndividual</i> du changement <i>RemoveClass</i>	101
4.14	Règle de réécriture du changement <i>PullUpClass</i>	103
5.1	Exemple de deux taxonomies.	109
5.2	Exemple de deux ontologies OWL.	110
5.3	Approche GROM de fusion d'ontologies.	111
5.4	Appariement d'ontologies avec SPO.	115
5.5	Règle de réécriture du changement <i>RenameObjectProperty</i>	115
5.6	Construction de l'ontologie commune.	116
5.7	Règle de réécriture de la construction de l'ontologie globale.	116
5.8	Règle de réécriture du changement <i>AddEquivalentClasses</i>	117
5.9	Règle de réécriture du changement <i>AddSubClass</i>	118
5.10	Résultat de fusion des ontologies illustrées dans la Figure 5.2.1 par l'approche GROM.	119
6.1	Relation entre les ontologies CCAIps.	123
6.2	Extrait de l'ontologie <i>RegionCCAIps</i>	124
6.3	Extrait de l'ontologie <i>EventCCAIps</i>	124
6.4	Extrait de l'ontologie <i>CompanyCCAIps</i>	125
6.5	Exemple 1 : ajout des classes et des axiomes de subsomption.	126
6.6	Exemple 2 : ajout d'individus et d'assertions.	126
6.7	Exemple 3 : diviser une classe.	127
6.8	Extrait de l'ontologie <i>EventCCAIps</i>	130
6.9	Toolchain pour l'évolution d'ontologies.	131
6.10	Interface graphique de l'outil AGG.	132
6.11	Processus de fusion d'ontologies GROM.	133
6.12	Comparaison des ontologies <i>Travel</i> (avant et après transformation) par l'outil SWOOP	134
6.13	Implémentation du changement <i>AddDisjointClasses(Meeting, Event)</i>	135

Liste des tableaux

2.1	Constructeurs du langage AL.	32
2.2	Les constructeurs OWL.	36
2.3	Les axiomes OWL.	36
2.4	Approches d'évolution d'ontologie.	52
2.5	Approches de fusion d'ontologies.	57
2.6	Positionnement et classification des approches d'évolution d'ontologies.	59
2.7	Positionnement et classification des approches de fusion d'ontologies.	60
4.1	Matrice de dépendance entre les changements élémentaires et les entités ontologiques.	95
4.2	Matrice de dépendance entre les changements ontologiques composés.	98
4.3	Matrice de dépendance des changements ontologiques complexes.	102
4.4	Formalisation des changements ontologiques selon [<i>Djedidi and Aufaure, 2010</i>] et l'approche proposée.	105
6.1	Exemple des changements dans les ontologies CCAIps.	127
6.2	Taille des LHS de certains changements ontologiques.	136
6.3	Fusion de certaines ontologies avec l'outil GROM.	138
A.1	Formalisation des changements ontologiques avec les grammaires de graphes . .	153

Chapitre 1

Introduction Générale

Sommaire

1.1 Introduction	19
1.2 Motivation et définition du problème	20
1.2.1 L'évolution d'ontologies	20
1.2.2 La fusion d'ontologies	22
1.2.3 Les grammaires de graphes	22
1.3 Approches proposées et contributions	23
1.4 Organisation du manuscrit	24
1.5 Publications	25

"Ce qui fait l'homme, c'est sa grande faculté d'adaptation."

Socrate

1.1 Introduction

"*Tout est changement, tout évolue, tout est en devenir, non pour ne plus être, mais pour devenir ce qui n'est pas encore*", c'est par cet aphorisme que le philosophe *Épictète* a parlé du besoin de changement et de l'importance de l'évolution dans notre vie. En effet, nous vivons dans un monde qui ne cesse de changer, de "progresser", de se développer, et il est naturel que l'Homme évolue également et cherche à trouver des solutions pour s'adapter aux changements qui l'entourent.

Étant une représentation formelle des connaissances humaines, les ontologies, elles aussi, font régulièrement l'objet de nombreux changements et ont ainsi besoin d'être constamment adaptées pour notamment pouvoir être réutilisées et répondre à de nouveaux besoins. C'est dans ce contexte que se placent les travaux de cette thèse. Nous nous sommes intéressés à

étudier la problématique de la réutilisation des ontologies et plus particulièrement leurs processus d'évolution et de fusion. Notre objectif est d'étudier l'existant, comprendre les enjeux et les verrous scientifiques de ce domaine et y trouver des solutions.

1.2 Motivation et définition du problème

Le travail de cette thèse s'inscrit dans le cadre du projet européen CCAAlps¹ (Creative Companies in Alpine Space). Ce projet vise à proposer une plateforme collaborative devant aider à mettre en relation les entreprises créatives et les régions de l'Espace Alpin (numéro de projet est 15-3-1-IT). Six partenaires de différents pays (la France, l'Allemagne, l'Italie, la Suisse, la Slovénie et l'Autriche) participent au projet. Ces différents partenaires organisent régulièrement des événements (des conférences, des bootcamps, etc.) et ont besoin de représenter et gérer leurs connaissances qui sont en constante évolution. Ainsi, afin de représenter ces connaissances et dans le but de créer un consensus entre les différents partenaires, l'idée directrice a été de faire appel aux ontologies.

En effet, les ontologies sont une représentation explicite des connaissances humaines en termes d'entités ontologiques qui sont les classes, les propriétés, les individus et les axiomes. Elles permettent de représenter des connaissances d'une manière formelle, compréhensible et accessible par des utilisateurs humains et/ou des programmes informatiques. Leur objectif principal est de créer un consensus entre une communauté d'utilisateurs en répondant aux problèmes d'hétérogénéité du vocabulaire et des ambiguïtés des sources de données d'un domaine donné. Les domaines modélisés subissent souvent des changements et les ontologies doivent ainsi prendre en compte les nouvelles connaissances qui deviennent pertinentes pour répondre aux attentes des utilisateurs. Dans notre travail, nous nous intéressons plus particulièrement à deux formes de changement des ontologies que sont l'évolution et la fusion.

1.2.1 L'évolution d'ontologies

Faire évoluer une ontologie consiste à modifier sa structure et/ou la peupler et ceci en appliquant un ensemble de changements (d'ajout, de suppression, de renommage, de fusion, etc.) affectant une ou plusieurs de ses entités (classe, propriété, axiome, individu, etc.). Il s'agit d'un processus incontournable dans le cycle de vie d'une ontologie qui devient de plus en plus demandé avec leur utilisation croissante. Nous citons à titre d'exemple que depuis Janvier 2010 jusqu'à 63 versions de l'ontologie Gene Ontology² ont été publiées à raison d'une version par mois. Ainsi, afin de définir et gérer le processus d'évolution, de très nombreux travaux et

1. www.ccalps.eu

2. geneontology.org/ontology-archive

approches ont été proposés dans la littérature affirmant l'importance et les enjeux de cette problématique. Parmi les principaux enjeux, nous citons : (1) la préservation de la consistance de l'ontologie évoluée ce qui nécessite la résolution des inconsistances pouvant avoir lieu lors du processus d'évolution et (2) la propagation des changements à la fois au niveau structurel (l'enrichissement d'ontologies) et au niveau assertionnel (le peuplement d'ontologies).

La littérature montre que certains travaux se sont focalisés sur l'étude de l'enrichissement d'ontologies [Klein, 2004]. D'autres ont étudié également le peuplement d'ontologies [Luong and Dieng-Kuntz, 2007, Djedidi and Aufaure, 2010]. La résolution des inconsistances a été relativement peu étudiée. En effet, certaines approches ont ignoré cet axe pour se concentrer sur d'autres problématiques, comme par exemple la gestion des versions des ontologies [Hartung et al., 2013]. D'autres travaux se sont focalisés plutôt sur l'identification des inconsistances [Gueffaz et al., 2012]. Seul quelques chercheurs se sont intéressés à trouver des solutions à la problématique de la résolution des inconsistances [Luong and Dieng-Kuntz, 2007, Djedidi and Aufaure, 2010, Javed et al., 2013, Pittet et al., 2014]. Ces approches admettent un processus a posteriori de leur traitement ce qui nécessite l'utilisation d'une ressource externe (tel qu'un raisonneur) afin de vérifier la consistance de l'ontologie évoluée. Cette stratégie de résolution n'est pas triviale puisqu'il est difficile de détecter l'origine des incohérences après l'application de plusieurs changements. Elle est aussi coûteuse en terme de ressources et de temps car elle exige, dans certains cas, de revenir en arrière et d'annuler tous les changements appliqués. Le travail de Jaziri et al. [2010] présente une approche préventive de résolution d'inconsistances. Toutefois, les inconsistances étudiées sont limitées et les changements se focalisent sur le niveau structurel des ontologies (pas de propagation de changement au niveau des individus et leurs assertions). Ainsi, pour synthétiser, nous pouvons dire que :

1. la préservation de la consistance de l'ontologie évoluée et la résolution des inconsistances sont encore des problématiques insuffisamment étudiées ;
2. les approches qui étudient les inconsistances les résolvent, généralement, d'une manière a posteriori en se basant sur des ressources externes (des raisonneurs ou/et des experts) ;
3. peu de travaux traitent le niveau assertionnel des ontologies et assurent la propagation des changements au niveau des individus.

Afin de répondre à ces problématiques, nous proposons l'utilisation des grammaires de graphes et les approches algébriques de transformations de graphes qui seront introduites après la présentation de notre deuxième axe de recherche, la fusion d'ontologies.

1.2.2 La fusion d'ontologies

Fusionner deux ou plusieurs ontologies consiste à créer une ontologie globale qui représente leur union. C'est un processus qui apparaît au cours de deux phases du cycle de vie d'une ontologie : lors de sa construction et/ou lors de son utilisation. En effet, une des stratégies adoptées pour la construction d'une ontologie est de fusionner certaines ontologies existantes. Cette stratégie est justifiée par la multitude d'ontologies représentant des domaines identiques ou connexes. Nous citons à titre d'exemple, que le domaine biomédical présente plus de 370 ontologies similaires et/ou complémentaires (par exemple, les ontologies Foundational Model of Anatomy (FMA), Systematized Nomenclature of Medicine-Clinical Terms (SNOMED-CT), National Cancer Institute Thesaurus (NCI), etc.). La fusion est également nécessaire pour enrichir les ontologies après leur construction afin d'intégrer de nouvelles connaissances.

Comme l'évolution, la fusion d'ontologies présente aussi de nombreux enjeux [Bellatreche et al., 2006, Klein, 2001]. Elle se compose essentiellement de deux grandes phases : 1) l'alignement des ontologies qui consiste à chercher les correspondances entre leurs concepts et 2) la fusion des ontologies en se basant sur l'alignement trouvé. Pour accomplir la première phase, plusieurs techniques de similarité ont été proposées dans la littérature [Pavel and Euzenat, 2013]. Quant à la phase de fusion d'ontologies, peu de travaux existent. Les plus importants sont ceux de Stumme and Maedche [2001], Noy and Musen [2003] et Raunich and Rahm [2014]. Cela dit, plusieurs questions restent en suspens et nous pouvons constater que :

1. il n'existe pas de consensus sur la stratégie de fusion "symétrique" ou bien "asymétrique" ;
2. peu de travaux se sont intéressés à la fusion d'ontologies lourdes (ontologies intégrant des axiomes) et se sont essentiellement focalisés sur de simples taxonomies ;
3. la résolution des conflits dans l'ontologie globale présente un grand défi.

A l'image des travaux effectués pour l'évolution d'ontologies, nous proposons l'utilisation des grammaires de graphes pour répondre aux problématiques de la fusion d'ontologies.

1.2.3 Les grammaires de graphes

Les grammaires de graphes, appelées aussi réécriture de graphes, sont un formalisme mathématique permettant de spécifier l'évolution des graphes [Rozenberg, 1999]. Elles sont utilisées dans plusieurs domaines de l'informatique telles que la modélisation de systèmes logiciels et la théorie des langages formels. Leur idée principale repose sur la modification de graphes via des règles de réécriture. Pour ceci, elles représentent le modèle étudié sous forme d'un graphe. Ensuite, et en se basant sur des concepts mathématiques, comme le concept Pushout (provenant de la théorie des catégories), décrivent formellement comment transformer

ce graphe d'un état à un autre tout en respectant un ensemble de conditions. En effet, les grammaires de graphes se distinguent par leurs conditions d'application notamment, les NAC (Negative Application Condition) qui interdisent l'application de tout changement ne satisfaisant pas un ensemble de contraintes.

Ainsi, un axe important de cette thèse est d'utiliser les grammaires de graphes dans le domaine des ontologies afin de trouver des solutions aux enjeux de leur réutilisation. L'idée consiste à représenter des ontologies par le formalisme des grammaires de graphes et profiter de ses concepts mathématiques rigoureux et ses conditions d'application pour formaliser les changements d'ontologies et préserver la consistance d'ontologies modifiées.

1.3 Approches proposées et contributions

Les principales contributions de cette thèse sont les suivantes :

1. La définition d'un nouveau formalisme permettant la représentation des ontologies et leurs changements par les grammaires de graphes typés.
2. La proposition d'une approche d'évolution d'ontologies traitant les inconsistances d'une manière a priori et permettant de préserver la consistance de l'ontologie évoluée. Nous nous sommes intéressés aux ontologies OWL (SHOIN(D)) et nous avons traité à la fois : (1) l'enrichissement d'ontologies en étudiant les différents changements (ajout, suppression, renommage, fusion, etc.) qui affectent les classes, les relations et les axiomes et (2) le peuplement d'ontologies en étudiant les changements qui affectent les individus et leurs assertions. Les changements ontologiques (élémentaires, composés et complexes) sont formalisés par des règles de réécriture et l'évolution des ontologies est assurée par l'approche algébrique Simple PushOut (SPO).
3. La proposition d'une approche asymétrique de fusion des ontologies OWL qui se décompose en trois étapes : (1) la recherche de similarité entre les concepts d'ontologies ; (2) la fusion des ontologies par l'approche algébrique SPO ; (3) l'adaptation de l'ontologie globale résultante à l'aide des règles de réécriture.
4. Le développement d'outils open source qui mettent en œuvre les différentes approches : (1) l'outil *OWLToGGX* permet de transformer une ontologie OWL au format GGX des grammaires de graphes ; (2) l'outil *GGXToOWL* permet de transformer une ontologie en format GGX au standard OWL ; (3) l'outil *GROM (Graph Rewriting for Ontology Merging)* permettant de fusionner deux ontologies selon notre formalisme défini ; (4) l'implémentation des changements ontologiques avec l'outil AGG (Attributed Graph Grammar).

1.4 Organisation du manuscrit

Ce manuscrit est composé de deux grandes parties. Dans la première partie, nous décrivons les différents états de l'art. Nous commençons par présenter les ontologies et les principaux travaux existants ayant contribué à leur processus d'évolution et de fusion. Ensuite, nous introduisons les grammaires de graphes, les approches de transformation et les travaux qui se sont intéressés à leur utilisation dans le domaine des ontologies. Dans la deuxième partie, nous présentons nos propositions. Nous détaillons notre approche d'évolution d'ontologies basée sur les grammaires de graphes typés et l'approche algébrique Simple PushOut (SPO). Nous introduisons également notre approche de fusion d'ontologies. Enfin, nous présentons le cadre applicatif des travaux de la thèse et les différents outils développés.

La suite de cette thèse est organisée comme suit :

Première partie : États de l'art

Le chapitre 2 présente les ontologies, leurs composants, leurs principaux langages de représentation et leur cycle de vie. Il décrit également les besoins et les enjeux de leur réutilisation, notamment pour les processus d'évolution et de fusion. Un état de l'art synthétise également les travaux traitant les problématiques d'évolution et de fusion d'ontologies. Il précise les avantages et les limites de chaque travail pour mieux positionner notre contribution.

Le chapitre 3 introduit les grammaires de graphes typés et les concepts nécessaires à la compréhension du formalisme proposé. Ce chapitre décrit également les approches utilisées pour la transformation de graphes et montre les caractéristiques de chacune. Enfin, il présente les travaux de la littérature qui se sont intéressés au couplage des ontologies et des grammaires de graphes.

Deuxième partie : Propositions

Le chapitre 4 présente notre approche d'évolution d'ontologies. Il décrit le modèle de transformation de graphes que nous proposons pour la représentation des ontologies et leurs changements. Ensuite, il détaille la formalisation des différents changements ontologiques étudiés (élémentaires, composés et complexes) tout en précisant comment assurer la préservation de la consistance de l'ontologie évoluée.

Le chapitre 5 décrit notre approche de fusion d'ontologies. Il détaille les différentes étapes menant à la construction de l'ontologie globale et à son adaptation. Il précise également comment éviter la redondance et réduire les conflits dans le résultat final de fusion en utilisant le formalisme de graphes défini dans le chapitre 4.

Le chapitre 6 présente le cadre applicatif de nos travaux ainsi que les outils développés pour valider les approches proposées. Il introduit également les ontologies du projet européen CCAIps et les autres ontologies benchmark utilisées pour tester nos méthodes.

Le chapitre 7 conclue la thèse en synthétisant les principales contributions et en présentant les perspectives envisagées : à court, à moyen et à long terme.

1.5 Publications

La liste suivante présente les publications concernant les travaux effectués dans cette thèse.

Revues internationales à comité de lecture

- [1] **Mahfoudh, M.**, G. Forestier, L. Thiry, and M. Hassenforder (2015), Algebraic graph transformations for formalizing ontology changes and evolving ontologies, *Knowledge-Based Systems (Impact Factor : 3.058)*, 73, 212–226.
- [2] Thiry, L., **M. Mahfoudh**, and M. Hassenforder (2014), A functional inference system for the web, *International Journal of Web Applications*, 6, 1–13.

Revue nationale à comité de lecture

- [3] **Mahfoudh, M.**, G. Forestier, L. Thiry, and M. Hassenforder (2015), Comment fusionner des ontologies avec la réécriture de graphes?, *Technique et Science Informatiques*, version étendue, en cours de lecture.

Conférences internationales à comité de lecture et avec actes

- [4] **Mahfoudh, M.**, L. Thiry, G. Forestier, and M. Hassenforder (2014), Algebraic graph transformations for merging ontologies, in *Model and Data Engineering, MEDI 2014*, pp. 154–168, Springer, Larnaca, Chypre.
- [5] **Mahfoudh, M.**, G. Forestier, L. Thiry, and M. Hassenforder (2013), Consistent ontologies evolution using graph grammars, in *Knowledge Science, Engineering and Management, KSEM 2013*, pp. 64–75, Springer, Dalian, China.

Conférences nationales à comité de lecture et avec actes

- [6] **Mahfoudh, M.**, L. Thiry, G. Forestier, and M. Hassenforder (2015), Une nouvelle formalisation des changements ontologiques composés et complexes, in *15èmes Journées Francophones Extraction et Gestion des Connaissances, EGC 2015*, pp. 263-274, RNTI(28), Luxembourg, Luxembourg (article long, 27 % d'acceptation).
- [7] **Mahfoudh, M.**, G. Forestier, L. Thiry, and M. Hassenforder (2014), Comment fusionner des ontologies avec la réécriture de graphes ?, in *Journées Francophones sur les ontologies, JFO 2014*, pp. 89-100, Hammamet, Tunisie.
- [8] **Mahfoudh, M.**, G. Forestier, L. Thiry, and M. Hassenforder (2014), Approche formelle de fusion d'ontologies à l'aide des grammaires de graphes typés, in *14èmes Journées Francophones Extraction et Gestion des Connaissances, EGC 2014*, pp. 565–568, RNTI(26), Rennes, France.
- [9] **Mahfoudh, M.**, L. Thiry, G. Forestier, and M. Hassenforder (2013), Adaptation consistante d'ontologies à l'aide des grammaires de graphe, in *Conférence d'Ingénierie des Connaissances, IC 2013*, Lille, France.

Première partie :
État de l'art

Chapitre 2

Réutilisation d'ontologies

Sommaire

2.1 Introduction	30
2.2 Langages de représentation d'ontologies	31
2.2.1 Logiques de description (LD)	32
2.2.2 Resource Description Framework (RDF)	34
2.2.3 Resource Description Framework Schema (RDFS)	34
2.2.4 Web Ontology Language (OWL)	35
2.3 Réutilisation d'ontologies	37
2.3.1 Cycle de vie des ontologies	37
2.3.2 Enjeux de la réutilisation d'ontologies	39
2.3.3 Cohérence d'ontologies	41
2.4 Évolution d'ontologies	43
2.4.1 Besoins de l'évolution	43
2.4.2 Classification des changements ontologiques	44
2.4.3 Approches d'évolution d'ontologies	45
2.5 Fusion d'ontologies	53
2.5.1 Besoins de fusion	53
2.5.2 Stratégies de fusion	54
2.5.3 Approches de fusion d'ontologies	55
2.6 Bilan et positionnement	58

"Les ontologies sont des objets vivants, et chaque étape de leur cycle de vie pose des problèmes de recherche."

Fabien L. Gandon

2.1 Introduction

Le mot ontologie trouve ses origines dans le domaine de la philosophie, plus précisément la branche de la métaphysique [Welty and Guarino, 2001]. Il a été proposé par Aristote et construit à partir de deux mots de racines grecques : 1) "onto" qui veut dire "ce qui existe" ou bien "l'existant" et 2) "logos" pour dire "la science" et "l'étude". D'où la signification de "l'étude de ce qui existe" et aussi "la science de l'être". Avec le développement de l'intelligence artificielle et le besoin indispensable de la représentation des connaissances, l'ingénierie des connaissances a choisi d'emprunter ce terme à la philosophie. Ceci a donné naissance à la première définition d'ontologie dans le domaine informatique : "*une ontologie définit les termes et les relations de base du vocabulaire d'un domaine ainsi que les règles qui indiquent comment combiner les termes et les relations de façon à pouvoir étendre le vocabulaire*" [Neches et al., 1991]. Plus tard, de nouvelles définitions ont été proposées dans la littérature. La plus citée est celle de [Gruber, 1993] : "*une ontologie est une spécification explicite d'une conceptualisation*". Ainsi, une ontologie est une représentation formelle des connaissances humaines en termes d'entités ontologiques qui sont :

- les *classes*, appelées aussi concepts, représentent pour un objet une notion ou une idée. Elles peuvent être la description d'une tâche, d'une fonction, d'une action, d'un processus de raisonnement, etc. [Gómez-Pérez, 1999] ;
- les *relations* traduisent les associations existantes entre les concepts d'une ontologie. Deux types de relations sont distinguées dans la littérature [Mhiri et al., 2006] :
 - les *relations conceptuelles* présentent les relations pouvant être trouvées dans un schéma conceptuel d'une base de données ;
 - les *relations sémantiques*, parfois appelées linguistiques, permettent d'apporter plus de sémantique aux modélisations élaborées [Baziz et al., 2003]. Elles aident à la compréhension du domaine modélisé et à éliminer certaines ambiguïtés linguistiques pouvant exister entre les concepts. Par exemple, la relation de synonymie relie deux concepts différents mais qui ont le même sens, comme par exemple, "personne" et "individu" ;
- les *axiomes* sont des assertions sous une forme logique permettant de : 1) définir la signification de certains concepts, 2) définir des restrictions sur des propriétés, 3) vérifier la cohérence logique d'une ontologie, 4) inférer de nouvelles connaissances, etc.
- les *instances*, appelées aussi individus, sont utilisées pour représenter des objets concrets des composants de l'ontologie et par conséquent du domaine du problème.

La Figure 2.1 illustre un exemple d'une ontologie composée de : (1) quatre classes ("Participant", "Événement", "Conférence" et "Bootcamps"), (2) une relation "participe" ("Participant", "Évènement"), (3) deux individus ("Mariem" et "KSEM") et (4) deux axiomes de subsomption

("Conférence" is-a "Evènement" et "Bootcamps" is-a "Conférence").

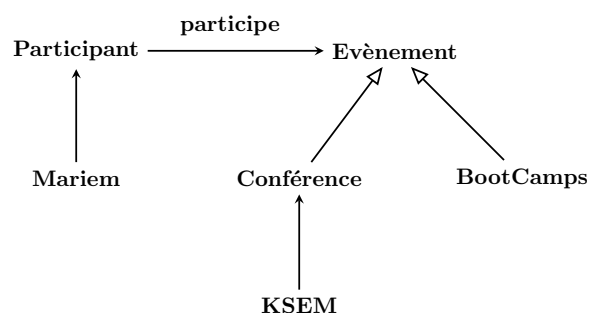


FIGURE 2.1 – Exemple d'une ontologie.

Les ontologies font régulièrement l'objet de nombreux changements et ont ainsi besoin d'être constamment adaptées et réutilisées. Leur réutilisation peut prendre différentes formes (évolution, alignement, fusion, etc.) et présente plusieurs verrous scientifiques.

L'objectif de ce chapitre est de présenter les principales approches traitant cette problématique. Nous commençons par introduire, les principaux langages utilisés pour la représentation des ontologies afin de se familiariser avec les spécificités de chacun. Par la suite, nous décrivons les enjeux de la réutilisation d'ontologies en parlant de leur cycle de vie et les différentes incohérences résultantes de leur changement. Enfin, nous détaillons certaines approches de la littérature ayant un lien avec les sujets de l'évolution et la fusion d'ontologies en précisant les avantages et les limites de chaque travail.

2.2 Langages de représentation d'ontologies

Les différents langages proposés dans la littérature pour la représentation des ontologies trouvent leurs fondements dans les logiques de description [Borgida, 1996, Calvanese et al., 2001, Baader et al., 2005] et les Frames [Minsky, 1975, Kifer et al., 1995]. Ces deux formalismes ont donné naissance à de nombreux langages, comme par exemple : Frame Logic (F-logic) [Kifer and Lausen, 1989], Knowledge Interchange Format (KIF) [Genesereth et al., 1992], Resource Description Framework (RDF) [Miller, 1998], Resource Description Framework Schema (RDFS) [McBride, 2004], DARPA Agent Markup Language (DAML-ONT) [McGuinness et al., 2003], Ontology Web Language (OWL) [McGuinness et al., 2004], Simple Knowledge Organisation System (SKOS) [Miles and Pérez-Agüera, 2007], etc.

Dans cette section nous nous intéressons à présenter le langage OWL, le standard proposé par le W3C¹ pour la représentation des ontologies, et aussi ses fondements : les logiques de description (LD) et les langages de web sémantique RDF et RDFS.

2.2.1 Logiques de description (LD)

Les logiques de description sont une famille de formalismes permettant la représentation formelle des connaissances d'un domaine particulier. Elles reposent sur trois entités : 1) les concepts correspondent à des classes d'individus ; 2) les rôles sont des relations (properties) entre ces individus et 3) les individus sont des exemples concrets des classes.

Deux composantes sont distinguées dans les logiques de description :

- la *Terminological box (Tbox)* qui définit la structure de l'ontologie (les concepts et les rôles) et les différentes axiomes.
- la *Assertional box (ABox)* qui précise les assertions sur les individus en spécifiant leur classes et leurs propriétés.

Les logiques de description s'appuient sur une base commune, *Attributive Language (AL)*, dont les constructeurs sont présentés par le Tableau 2.1. Ces derniers sont capables de représenter des connaissances avec une expressivité simple, voir l'exemple de la Figure 2.2 qui reproduit l'ontologie décrite sur la Figure 2.1) avec le langage AL.

Constructeur	Syntaxe
concept atomique	A
concept universel	\top
concept impossible	\perp
négation atomique	$\neg A$
intersection de concepts	$C \sqcap D$
restriction de valeur	$\forall R.C$
quantification existentielle limitée	$\exists R.\top$

TABLE 2.1: Constructeurs du langage AL.

Afin de définir des propriétés et des axiomes beaucoup plus expressifs, le langage AL a été enrichi par plusieurs extensions formant ce qu'on appelle la famille des logiques de description (voir Figure 2.3).

Nous distinguons, ainsi :

- le langage $ALC = (AL + \neg C)$ qui constitue la base de tous les langages de description

1. <http://www.w3.org>

TBox
 $Bootcamps \sqsubseteq Evènement$
 $Conférence \sqsubseteq Evènement$
 $\exists participe \sqsubseteq Participant$
 $\top \sqsubseteq \forall participe.Evènement$

ABox
 $Conférence (KSEM)$
 $Participant (Mariem)$

FIGURE 2.2 – Exemple d’une ontologie représentée avec le langage AL.

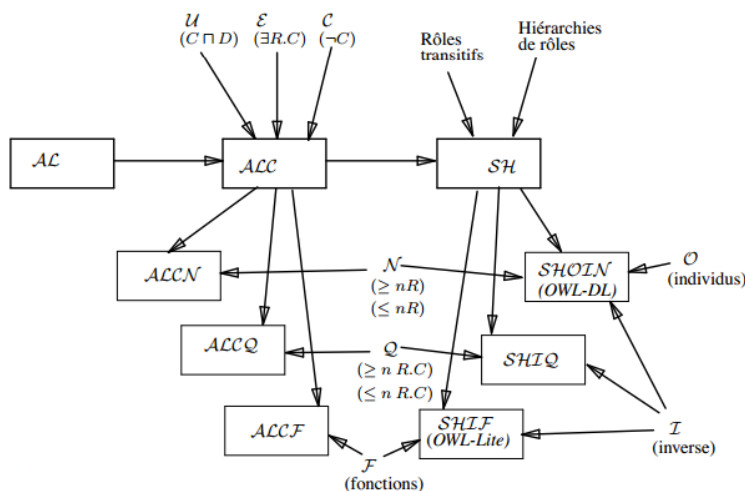


FIGURE 2.3 – Famille des logiques de description [Gagnon, 2007].

- expressifs. Il permet d’exprimer : 1) l’union entre concepts ($C_1 \sqcup C_2$), 2) le complément ($\neg C$) qui exprime la négation d’une classe, 3) la quantification existentielle complète ($\exists R.C$) qui permet de spécifier qu’une entité doit avoir au moins une relation avec un objet tout en précisant la classe de cet objet (ex. la spécification $\exists participe.Bootcamps$ définit la classe de tous les gens qui ont participé au moins à un "Bootcamps") ;
- la famille $SH = (ALC + H + Tr(R))$ vise à enrichir les rôles (relations). Elle permet, par exemple, de spécifier la transitivité d’un rôle ($Tr(R)$) et aussi la hiérarchie entre les rôles (H) qui exprime qu’une relation est une sous propriété d’une autre relation ($R_1 \sqsubseteq R_2$) ;
 - le langage $SHIF = (SH + I + F)$ inclut les constructeurs de SH et en ajoute d’autres pour définir l’inverse d’un rôle (I) et des fonctions (F) permettant de spécifier qu’un rôle est une fonction, c’est-à-dire, qu’aucune entité ne peut être reliée à plus d’une autre entité par cette relation.

- le langage *SHOIN* = (*SH* + *O* + *I* + *N*) permet d'exprimer des restrictions de cardinalité nommées *N* ($\leq n.R$ et $\geq n.R$) et de l'énumération (*O*) qui définit une classe à partir d'une liste d'individus, ex. *Saisons* $\equiv \{Automne, Hiver, Printemps, Été\}$.

2.2.2 Resource Description Framework (RDF)

RDF (Resource Description Framework) est un langage du web sémantique ayant une sémantique simplifiée. Il permet de représenter formellement des ressources publiées sur le web et les relations entre elles. Les données RDF sont décrites par des triplets de la forme $\langle \textit{ sujet}, \textit{ prédicat}, \textit{ objet} \rangle$ où :

- le *sujet* représente la ressource à décrire ;
- le *prédicat* représente un type de propriété applicable à cette ressource ;
- l'*objet* représente la valeur de la propriété.

Le sujet, le prédicat et, souvent, l'objet sont identifiés par des URI (Uniform Resource Identifier) ou aussi par des IRI (Internationalized Resource Identifier) [Dürst, 2001] assurant l'accessibilité et le partage des ressources décrites.

Un triplet RDF est représenté sous forme d'un graphe orienté et étiqueté. Le sujet et l'objet sont représentés par des nœuds (sommets) et le prédicat est illustré par un arc reliant le sujet à l'objet. La Figure 2.4 présente un exemple d'un graphe RDF décrivant une partie d'un site web. La ressource "<http://www.mariem-mahfoudh.info>" représente le sujet. "a-été-crée-par" correspond au prédicat et "Mariem-Mahfoudh" représente l'objet.

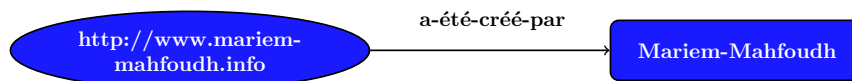


FIGURE 2.4 – Exemple d'un graphe RDF

2.2.3 Resource Description Framework Schema (RDFS)

RDFS est un langage formel qui a été principalement proposé pour étendre la sémantique insuffisante du langage RDF. Il permet de déclarer des classes de ressources (`rdfs: class`) et des liens de hiérarchies entre elles (`rdfs: subclassOf`). Il a également la capacité de définir des propriétés entre les classes (`rdfs: property`), préciser leurs membres (`rdfs: domain` et `rdfs: range`) et décrire leurs hiérarchies (`rdfs: subPropertyOf`). Ces caractéristiques permettent à RDFS de représenter des taxonomies ou des ontologies légères. À noter qu'une

taxonomie est une classification par sujet qui organise les termes d'un vocabulaire dans une hiérarchie [Garshol, 2004]. Alors que, les *ontologies légères (light-weight ontologies)* comprennent les concepts, les taxonomies de concepts, les relations entre les concepts et les propriétés qui décrivent les concepts [Giunchiglia and Zaihrayeu, 2009]. Malgré sa sémantique étendue, RDFS souffre de certaines limites. Il ne permet pas, par exemple, d'exprimer les caractéristiques des propriétés (la transitivité, la symétrie, etc.) ou bien de définir des cardinalités. Pour remédier à ces limites, le langage OWL a été proposé.

2.2.4 Web Ontology Language (OWL)

OWL est le standard actuellement proposé par le W3C pour représenter les ontologies. C'est un langage du web sémantique basé sur RDFS et les logiques de description (voir certains constructeurs et axiomes présentés, respectivement, dans les tableaux 2.2 et 2.3). OWL est un langage expressif qui couvre plusieurs fonctionnalités. Il permet d'exprimer la disjonction entre classes (`owl:disjointWith`), l'équivalence (`owl:equivalentOf`), des restrictions (`owl:Restriction`), des cardinalités (`owl:cardinality`, `owl:minCardinality`, `owl:maxCardinality`), des classes énumérées (`owl:oneOf`), etc.

Le langage OWL possède deux versions OWL1 et OWL2. OWL1 a été recommandé depuis 2004 et offre trois sous-langages d'expressivité croissante :

- *OWL Lite* est une version d'OWL aux fonctionnalités réduites (cardinalité restreinte à 0 ou 1, pas de négation, etc.) mais qui restent suffisantes pour bien des usages, comme la construction des ontologies légères. Cette variante correspond au langage SHIF de la logique de description. Plus précisément, le SHIF(D) où D correspond au Dataproperty qui est un constructeur permettant de relier un individu à un littéral.
- *OWL DL* est un langage expressif dont les procédures d'inférences sont complètes, c'est-à-dire, toutes les inférences sont calculables. Il correspond au langage SHOIN(D) et souvent utilisé pour représenter des *ontologies lourdes (heavy-weight ontologies)*. À noter que les ontologies lourdes sont des "*ontologies intégrant des axiomes permettant de fixer toute la sémantique du domaine considéré*" [Fürst and Trichet, 2006];
- *OWL Full* est la variante la plus complexe d'OWL qui offre une expressivité maximale, mais ne propose aucune garantie sur la complétude et sur la terminaison des procédures d'inférence.

Constructeur OWL	Syntaxe LD
<code>intersectionOf (C₁, C₂, ...)</code>	$C_1 \sqcap C_2$
<code>unionOf (C₁, C₂, ...)</code>	$C_1 \sqcup C_2$
<code>complementOf(C)</code>	$\neg C$

oneOf(I_1, I_2, \dots)	$\{I_1, I_2, \dots\}$
Restriction(P allValuesFrom(C))	$\forall P.C$
Restriction(P someValuesFrom(C))	$\exists P.C$
Restriction(P hasValue(I))	$P : I$
Restriction(P cardinality(n))	$= nP$
Restriction(P minCardinality(n))	$\leq nP$
Restriction(P maxcardinality(n))	$\geq nP$

TABLE 2.2: Les constructeurs OWL.

Constructeur OWL	Syntaxe LD
Axiomes sur les classes	
subClassOf (C_1, C_2)	$C_1 \sqsubseteq C_2$
equivalentClasses (C_1, \dots, C_n)	$C_1 \equiv \dots \equiv C_n$
disjointWith (C_1, \dots, C_2)	$C_1 \sqsubseteq \neg C_2$
Axiomes sur les propriétés	
subPropertyOf (P_1, P_2)	$P_1 \sqsubseteq P_2$
equivalentProperties (P_1, \dots, P_n)	$P_1 \equiv \dots \equiv P_n$
inverseOf (P_1, P_2)	$P_1 \equiv P_2^-$
symetricProperty(P)	$P \equiv P^-$
functionalProperty (P)	$\top \sqsubseteq \leq 1P$
inverseFunctionalProperty (P)	$\top \sqsubseteq \leq 1P^-$
transitiveProperty(P)	$Tr(P)$
Axiomes sur les individus	
sameAs(I_1, \dots, I_n)	$I_1 = \dots = I_n$
differentFrom(I_1, \dots, I_n)	$I_1 \neq \dots \neq I_n$

TABLE 2.3: Les axiomes OWL.

La syntaxe LD peut être représentée par le format RDF/XML. Ainsi, les classes sont définies par la balise `<owl:Class>`. Les propriétés sont représentées par deux balises : 1) `<owl:ObjectProperty>` permet de relier un objet (classe) à un autre objet (classe) et 2) `<owl:DatatypeProperty>` permet de relier un objet (classe) à un type de donné (datatype). Les types des individus sont spécifiés par `<rdf:type>`. La Figure 2.5 illustre la représentation OWL (avec la syntaxe RDF/XML) de l'ontologie présentée dans la Figure 2.1 (Section 2.1) et la Figure 2.2 (Section 2.2.1).

```
        /** Classes **/  
<owl:Class rdf:about="#Evènement"/>  
<owl:Class rdf:about="#Participant"/>  
<owl:Class rdf:about="#Conférence">  
    <rdfs:subClassOf rdf:resource="#Evènement"/>  
</owl:Class>  
<owl:Class rdf:about="#Bootcamps">  
    <rdfs:subClassOf rdf:resource="#Evènement"/>  
</owl:Class>  
  
        /** ObjectProperties **/  
<owl:ObjectProperty rdf:about="#participe">  
    <rdfs:range rdf:resource="#Evènement"/>  
    <rdfs:domain rdf:resource="#Participant"/>  
</owl:ObjectProperty>  
  
        /** Individuals **/  
<owl:Thing rdf:about="#KSEM">  
    <rdf:type rdf:resource="#Conférence"/>  
</owl:Thing>  
<owl:Thing rdf:about="#Mariem">  
    <rdf:type rdf:resource="#Participant"/>  
</owl:Thing>
```

FIGURE 2.5 – Exemple d'une ontologie OWL représentée avec la syntaxe RDF/XML.

2.3 Réutilisation d'ontologies

Étant des objets dynamiques et vivants [Gandon *et al.*, 2008], les ontologies subissent constamment des changements. Leur cycle de vie ne se termine pas avec leur construction mais continue pour assurer leur adaptation et réutilisation. Cette réutilisation peut prendre différentes formes comme par exemple l'évolution ou la fusion et nécessite la préservation de la consistance de l'ontologie après son changement. Dans ce qui suit, nous présentons le cycle de vie des ontologies et ses différentes activités. Ensuite, nous introduisons les concepts liés à la consistance d'une ontologie et les incohérences pouvant être identifiées lors du processus de réutilisation.

2.3.1 Cycle de vie des ontologies

Le cycle de vie d'une ontologie rassemble plusieurs activités allant de l'identification des besoins pour la construction de l'ontologie jusqu'à sa maintenance. La Figure 2.6 présente les activités les plus importantes identifiées dans les travaux de *Fernández-López et al.* [1997],

Dieng et al. [2001] et de Gandon [2002].

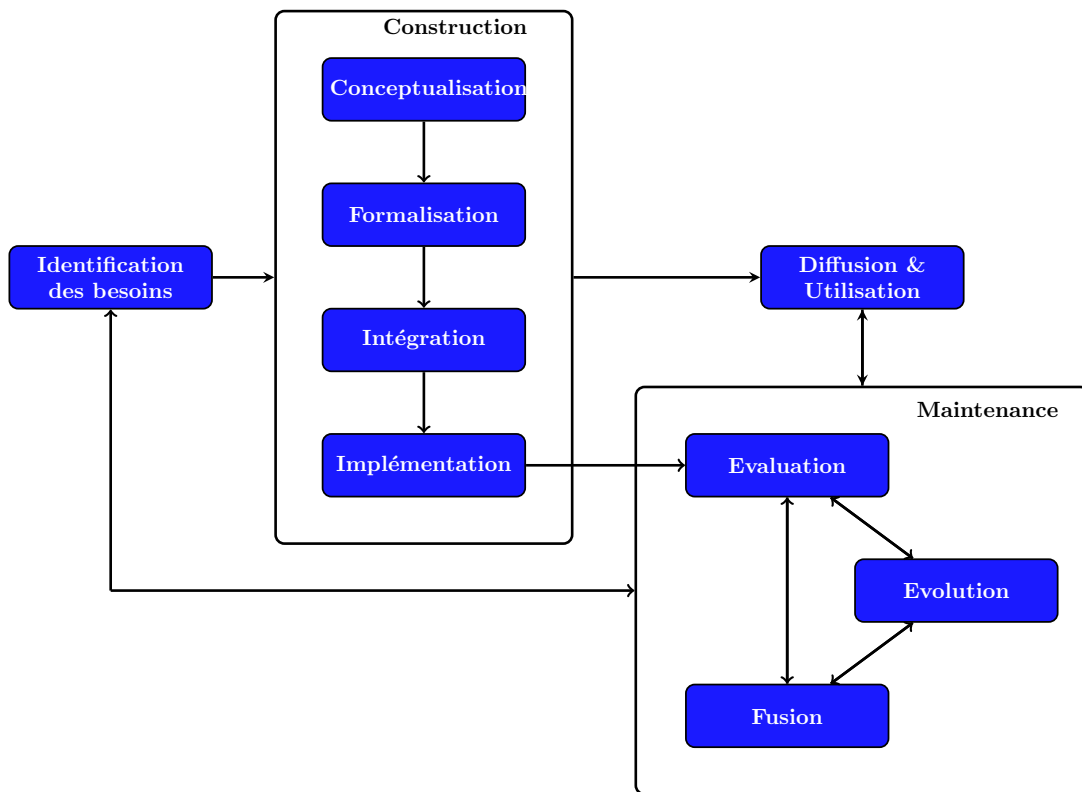


FIGURE 2.6 – Cycle de vie d'une ontologie.

Ainsi, nous distinguons les activités suivantes :

1. *l'identification ou la détection des besoins* permet d'identifier les utilisateurs et leurs besoins. Elle définit aussi les scénarios d'utilisation pour justifier le développement de l'ontologie et cherche les connaissances manquantes afin de trouver un consensus nécessaire pour la phase de construction.
2. la *construction* consiste à créer l'ontologie. Elle est constituée des étapes suivantes :
 - la *conceptualisation* permet de représenter le domaine de discours par des entités ontologiques qui sont les classes, les relations, les axiomes et les individus (voir Section 2.1). Elle nécessite une bonne structuration des entités et un recours aux dictionnaires afin d'éviter toutes ambiguïtés liées aux termes utilisés et/ou aux domaines étudiés.
 - la *formalisation* est la représentation de la conceptualisation définie par un langage

formel (voir Section 2.2).

- *l'intégration* consiste à faire appel à des ontologies existantes et les intégrer dans l'ontologie créée dans le but de réduire l'effort et le coût de mise en œuvre. Ceci n'est pas toujours évident puisque la réutilisation d'ontologies présente plusieurs enjeux (voir Section 2.3.2).
- *l'implémentation* consiste à créer l'ontologie physiquement.

3. la *maintenance* consiste à vérifier la qualité de l'ontologie construite et à l'adapter pour prendre en compte de nouveaux besoins. Cette activité englobe :

- *l'évaluation* qui permet de vérifier la qualité de l'ontologie et sa capacité de répondre aux attentes des utilisateurs. Elle utilise pour ceci des tests et des requêtes, et se base essentiellement sur un ensemble de critères (métriques) qui sont : la clarté, l'accessibilité, la cohérence et la consistance de l'ontologie, la capacité d'inférence, etc. Une description détaillée de ces métriques peut être trouvée dans [Uschold and Gruninger, 1996, Noy and Hafner, 1997, Djedidi, 2009].
- *l'évolution* consiste à modifier la structure et/ou les instances de l'ontologie. Elle vise à réviser l'ontologie et la faire évoluer pour répondre à de nouveaux besoins (plus de détails dans la Section 2.4).
- la *fusion* consiste à fusionner deux ontologies, ou plus, pour créer une nouvelle ontologie (voir Section 5.4). En effet, après la phase de construction et de vérification l'ontologie peut faire partie du processus de construction d'autres ontologies (activité d'intégration).

4. la *diffusion* et *l'utilisation* assurent le déploiement de l'ontologie et son utilisation par les utilisateurs. Ce sont les activités qui suivent la construction de l'ontologie et sa maintenance.

2.3.2 Enjeux de la réutilisation d'ontologies

La réutilisation d'ontologies est une activité fondamentale dans le cycle de vie d'une ontologie. Elle est présentée à la fois, dans la phase de maintenance (évolution et fusion) et aussi dans la phase de construction (intégration). Cependant, adapter ou réutiliser une ontologie est un processus non trivial présentant différents enjeux. Dans ce qui suit, nous distinguons certains enjeux pour l'évolution et la fusion. D'autres seront décrits au fur et à mesure de ce chapitre.

Enjeux de l'évolution d'ontologies

- (E1. *Identification des besoins d'évolution*). Les besoins d'évolution d'ontologies doivent être identifiés correctement afin de formuler les bons changements ontologiques. La question principale qui se pose ici : est-t-il mieux d'adopter une stratégie manuelle et faire intervenir l'utilisateur en lui demandant d'exprimer ses besoins d'évolution ? ou bien automatiser le processus en identifiant automatiquement ces besoins, par exemple par l'étude des profils d'utilisateurs ou l'analyse de corpus de données, etc.
- (E2. *Qualité de l'ontologie initiale*). L'ontologie à faire évoluer doit être dans un état consistant, respecter les contraintes de son langage de représentation et ne contenir aucun axiomes contradictoires (une description détaillée de la cohérence et la consistance de l'ontologie sera présentée dans la Section 2.3.3).
- (E3. *Formalisation des changements*). Les changements ontologiques doivent être décrits et représentés par un langage formel.
- (E4. *Préservation de la consistance de l'ontologie évoluée*). L'application des changements ontologiques ne doit pas altérer la consistance de l'ontologie évoluée.

Enjeux de la fusion d'ontologies

La fusion d'ontologies possède les mêmes enjeux que l'évolution et en ajoute d'autres. En se basant sur les travaux de [Klein, 2001, Bellatreche et al., 2006], nous identifions :

- (E1. *Diversité de formalismes*). Les ontologies peuvent être représentées par différents langages (RDF(S), KAON, OWL, etc). Ceci pose des vrais problèmes lors de la recherche des correspondances entre elles et bien entendu lors de la phase de fusion. Plusieurs questions se posent dans ce contexte. Faut-il transformer les deux ontologies en un langage unifié ? Choisit-on plutôt le langage le plus expressif des deux ontologies et transformer l'autre ? Existe-t-il des outils capables de réaliser ces transformations sans perte d'informations ?, etc.
- (E2. *Diversité des domaines*). Les ontologies à fusionner peuvent être issues des domaines différents. Ceci pose des questions sur le sens des termes utilisés dans chacune. Par exemple, le terme "fils" dans le domaine génétique peut avoir un sens différent dans le domaine du textile. Alors, comment peut-on identifier la nature des liens à établir entre les ontologies ? S'agit-il des relations de synonymie ou plutôt polysémie ?, etc. A noter que la relation de polysémie, dite aussi d'homonymie, est une relation liant deux concepts qui ont la même syntaxe mais des sens différents.
- (E3. *Langue des ontologies*). Les ontologies peuvent être exprimées dans des langues différentes (anglais, français, arabe, etc). Ceci pose des questions sur la manière de leur

intégration et la disponibilité des dictionnaires et ressources linguistiques suffisantes pour leur traduction.

2.3.3 Cohérence d'ontologies

La cohérence et la consistance d'une ontologie sont deux caractéristiques fondamentales définissant sa qualité. Elles sont utilisées dans de nombreux travaux pour désigner la même notion "l'absence de contradiction entre les concepts d'ontologies" [Djedidi and Aupaure, 2010]. Cependant, une différence est souvent identifiée pour les ontologies OWL [Flouris et al.]. Dans ce qui suit, nous introduisons ces deux notions. Nous présentons les différentes incohérences/inconsistances pouvant être détectées lors du changement des ontologies ainsi que les approches adoptées pour les vérifier.

Une ontologie est dite "cohérente par rapport à son modèle, si et seulement si, elle satisfait les contraintes spécifiées par ce modèle" [Stojanovic, 2004]. Elle est donc incohérente si elle contient au moins un concept insatisfaisable, c'est-à-dire, un concept qui n'a pas d'interprétation et ne peut accepter donc aucune instance. Ainsi, la cohérence d'une ontologie est déterminée par l'absence de contradictions logiques entre les représentations qu'elle contient et elle est liée à la TBox. En revanche, la consistance de l'ontologie est liée à l'ABox et consiste à que tous les individus de l'ontologie soient valides.

La Figure 2.7 présente deux exemples d'ontologies. La première (a) est une ontologie cohérente mais inconsistante. Deux concepts disjoints ne peuvent pas partager des individus communs. La Figure (b) présente une ontologie consistante mais incohérente. Deux classes disjointes ne peuvent pas avoir des sous-classes communes. La classe C3 est une classe insatisfaisable.

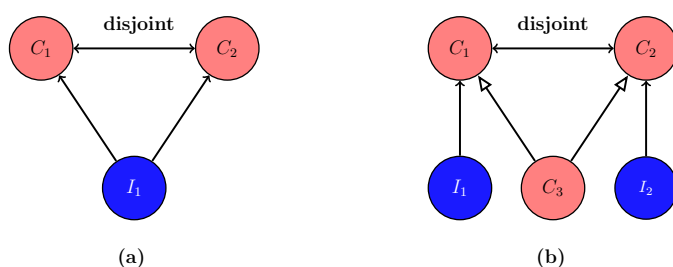


FIGURE 2.7 – Exemples d'inconsistances et d'incohérences.

Types de cohérence

Plusieurs types de cohérence ont été évoqués dans la littérature [Haase and Stojanovic, 2005, Djedidi, 2009]. Nous distinguons :

- la *cohérence métier*, appelée aussi cohérence de domaine, représente la cohérence l'ontologie par rapport au domaine d'étude. Chaque domaine peut être représenté par un ensemble de concepts clés. Ces concepts ne doivent pas être supprimés (mais peuvent être remplacés par d'autres concepts équivalents et représentatifs du domaine) de l'ontologie afin de conserver sa cohérence par rapport au domaine modélisé.
- la *cohérence interne* relative à la sémantique de l'ontologie et à sa structure conceptuelle. Deux sous types sont distingués :
 - *cohérence conceptuelle* qui se réfère aux règles structurelles et contraintes du langage de représentation de l'ontologie. Ex. inexistence de concepts isolés.
 - *cohérence logique*, dite aussi cohérence sémantique, qui se réfère à la cohérence logique de l'ontologie dans le sens où elle ne doit pas comporter des contradictions logiques. Ex. ne pas avoir deux relations contradictoires entre deux concepts.

Vérification de la cohérence

Deux stratégies sont adoptées pour la vérification de la cohérence et la consistance de l'ontologie :

- une *approche a posteriori* qui nécessite l'application de tous les changements pour vérifier par la suite la consistance globale de l'ontologie. Il faut donc appliquer, en premier lieu, les différents changements souhaités. Ensuite, pour détecter les incohérences logiques, la vérification est réalisée par des raisonneurs, comme par exemple Pellet [Sirin et al., 2007], HermiT [Motik et al., 2009], Fact [Tsarkov and Horrocks, 2006], etc. Une fois détectée, une liste des incohérences sera affichée à l'ingénieur d'ontologie (appelé aussi ontologiste) et différentes possibilités de résolution, appelées aussi *stratégies de résolution* sont générées. Cette approche n'est pas triviale puisqu'il est difficile de détecter l'origine des incohérences après l'application de plusieurs changements. Elle est aussi coûteuse en terme de ressources et de temps car elle fait appel à une ressource externe : les raisonneurs. De plus, elle exige, dans certains cas, de revenir en arrière et d'annuler tous les changements appliqués.
- une *approche a priori* qui exige la satisfaction d'un ensemble de pré-conditions pour pouvoir appliquer les changements. Ces conditions assurent la préservation de l'ontologie et nécessitent, par ailleurs, que l'ontologie soit cohérente dans son état initial. La résolution d'inconsistance est réalisée d'une manière procédurale : à chaque changement, les

pré-conditions sont vérifiées puis les résolutions d'incohérences sont générées.

2.4 Évolution d'ontologies

L'évolution d'ontologie est "*l'adaptation – dans le temps – d'une ontologie aux besoins de changement et la propagation cohérente des changements aux artefacts dépendants*" [Stojanovic, 2004]. Elle se traduit par une modification d'une ou plusieurs entités ontologiques (classe, propriété, axiome, individus, etc.) et est appelée *changement ontologique*. L'évolution peut être à deux niveaux [Khattak et al., 2013a] : structurel et assertionnel. Elle peut ainsi viser la modification de la structure de l'ontologie (ex. ajout de classe, ajout de propriété) et on parle dans ce cas de *l'enrichissement d'ontologie (ontology enrichment)*. Elle peut viser également la création d'individus et des assertions sur celles-ci et on parle alors de *peuplement d'ontologie (ontology population)*.

Cette section s'intéresse à introduire l'évolution d'ontologie et à répondre aux questions suivantes : (1) Pourquoi les ontologies ont besoin d'évoluer ? (2) Comment exprimer cette évolution ? (3) Quelles sont les approches proposées dans la littérature ayant traité cet axe de recherche et quelles sont leurs contributions et leurs limites ?

2.4.1 Besoins de l'évolution

Avec la prolifération et la large utilisation des ontologies, leur évolution est devenue un processus incontournable dans leur cycle de vie. Ce besoin d'évolution peut être expliqué par de nombreuses raisons. Dans ce qui suit, nous présentons les plus importantes :

- *Évolution du domaine modélisé*. Un besoin de changement d'ontologie peut provenir de la dynamique du domaine modélisé. En effet, les ontologies représentent souvent des domaines dynamiques et ont besoin à cet égard de s'adapter pour prendre en compte des nouvelles connaissances qui deviennent pertinentes dans le domaine ;
- *Changement des besoins d'utilisateurs*. Les ontologies doivent toujours répondre aux attentes d'utilisateurs. Elles ont besoin, par conséquent, d'être à jour et d'évoluer d'une manière permanente pour suivre les nouvelles exigences.
- *Changement de la conceptualisation de l'ontologie*. Les ontologies réfèrent à une vue du monde, une représentation d'un domaine de discours en termes d'entités ontologiques. Elles doivent, ainsi, évoluer pour répondre aux différents points de vue sur la conceptualisation qu'elles spécifient.
- *Mettre à jour une ontologie dans un environnement distribué*. Si une ontologie appartient à un environnement distribué, alors elle doit toujours être en harmonie avec les ontologies avec lesquelles elle est reliée et prendre en compte les modifications qu'elles subissent.

2.4.2 Classification des changements ontologiques

Plusieurs changements ontologiques peuvent exprimer les besoins d'évolution. Deux classifications importantes ont été proposées dans la littérature.

Classification de Klein [2004] a été proposée pour classifier les changements d'ontologies OWL. Elle distingue deux types de changements :

- les changements élémentaires qui modifient une seule caractéristique du modèle de l'ontologie ;
- les changements composés qui affectent plusieurs entités de l'ontologie.

Classification de Stojanovic [2004] a été proposée pour classifier les changements d'ontologies KAON (KAarlsruhe ONtology). Elle distingue trois types de changements selon leur impact sur les entités de l'ontologie (voir Figure 4.6) :

- un *changement élémentaire* représente une opération primitive et non décomposable qui affecte une seule entité de l'ontologie (ex. ajouter un lien de subsomption entre deux classes) ;
- un *changement composé* affecte une entité ontologique et ses voisins (ex. suppression d'une classe). Ce type de changement nécessite des changements additionnels permettant de préserver la consistance de l'ontologie, appelés changements dérivés ;
- un *changement complexe* exprime un enchaînement de plusieurs changements élémentaires et composés. Il touche des entités qui ne sont pas obligatoirement voisines (ex. déplacer un concept qui permet d'attacher un concept à un autre qui n'est pas nécessairement son fils ou son père direct dans la hiérarchie de l'ontologie).

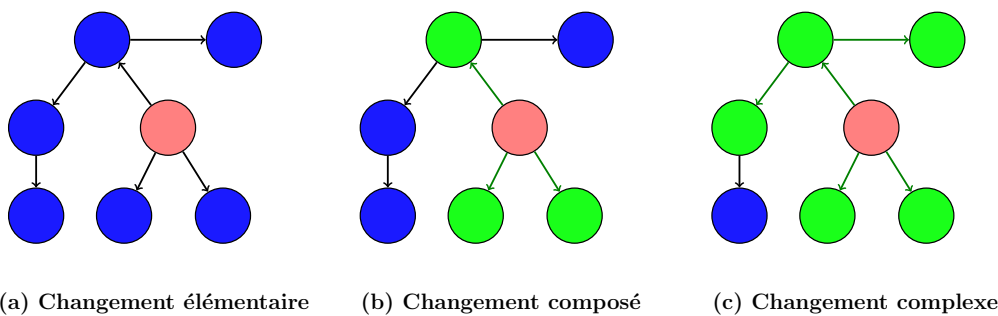


FIGURE 2.8 – L'impact des changements ontologiques sur les entités de l'ontologie.

2.4.3 Approches d'évolution d'ontologies

De nombreuses approches et outils ont été proposés pour aider à gérer l'évolution des ontologies. Dans ce qui suit, nous présentons les principaux travaux.

Stojanovic et al. [2002] ont proposé une approche globale d'évolution d'ontologies KAON composée de six phases [Stojanovic, 2004] :

1. *L'identification des changements* permet de déterminer les concepts d'ontologies susceptibles d'être évolués. Deux moyens sont adoptés pour cette identification : a) *les changements dirigés par l'usage* qui se basent sur l'analyse du comportement des utilisateurs et la traçabilité de leurs requêtes ; b) *les changements dirigés par les données* qui sont dérivés des versions d'ontologies.
2. *La représentation des changements* permet de représenter les changements ontologiques simples et composés selon les spécifications du langage KAON.
3. *La sémantique des changements* permet de résoudre les conflits des changements ontologiques afin de préserver la cohérence de l'ontologie.
4. *La propagation des changements* consiste à transmettre les changements de l'ontologie aux artefacts dépendants (ontologies réutilisées par l'ontologie évoluée ou des applications distribuées) afin de préserver la cohérence globale.
5. *L'implémentation des changements* consiste à implémenter physiquement les changements ontologiques avec l'intervention de l'ingénieur d'ontologie qui les prouve et les applique.
6. *La validation des changements* valide l'application finale des changements.

Le travail de *Stojanovic et al. [2002]* est considéré parmi les premiers travaux traitant la problématique d'évolution d'ontologie. Il a introduit les phases du processus de l'évolution et a proposé un outil fonctionnel nommé KAON.

La principale limite de ce travail est qu'il traite essentiellement les ontologies KAON. Ceci restreint le champ d'application de l'approche étant donné que, de nos jours, la plupart des ontologies sont représentées par les langages RDFS et OWL.

Klein and Noy [2003] ont défini une approche de gestion des changements pour les ontologies distribuées. L'approche permet d'identifier la différence entre les versions de l'ontologie évoluée et de spécifier des relations entre ces versions [Klein, 2004]. La gestion des versions des ontologies est réalisée à l'aide du framework ONTOVIEW qui permet également de sauvegarder la traçabilité des changements ontologiques.

Les auteurs ont traité la problématique des inconsistances et ont proposé des stratégies pour les résoudre. Cependant, il est important de noter que ce travail est focalisé sur "l'enrichissement d'ontologies" et ignore le "peuplement d'ontologies". Autrement dit, aucun changement n'est proposé pour traiter les individus.

Luong and Dieng-Kuntz [2007] ont proposé une approche d'évolution d'ontologies dans le cadre du web sémantique d'entreprise (WSE) qui s'intéresse à faciliter et manipuler les connaissances organisationnelles. Le travail traite à la fois l'évolution d'ontologies et de l'annotation sémantique optant ainsi deux scénarios : avec et sans trace de changements ontologiques. Les traces des changements sont sauvegardées en format RDF et utilisées pour propager les changements de l'ontologie aux annotations qui la référencent. La propagation est réalisée selon des stratégies d'évolution définissant les types de changements et les règles de cohérences étudiés. Afin de valider l'approche, les auteurs proposent un outil CoSWEM (Corporate Semantic Web Evolution Management) permettant de guider l'utilisateur dans le processus d'évolution.

Il est intéressant de noter que ce travail propose différents changements ontologiques du type simples et composés. Cependant, et étant donné que seules les ontologies RDFS ont été étudiées, plusieurs changements ne sont pas traités, comme par exemple la gestion des restrictions, les cardinalités, etc.

Djedidi and Aufaure [2010] ont proposé une approche d'évolution d'ontologies Onto-Evoal (Ontology Evolution-Evaluation) basée sur une modélisation de patrons de gestion de changement, CMP (Change Management Patterns). Les patrons spécifient des classes de changements (Change Patterns), des classes d'incohérences (Inconsistency Patterns) et des classes d'alternatives de résolution (Alternative Patterns). Ils permettent de gérer le processus d'évolution tout en maintenant la cohérence de l'ontologie évoluée. L'approche Onto-Evoal traite les ontologies OWL-DL et elle est composée de trois phases principales :

1. La *spécification des changements* qui permet de décrire formellement la sémantique du changement demandé par l'utilisateur. Elle est guidée par l'instanciation du patron de changements.
2. L'*analyse des changements* qui consiste à analyser l'impact du changement en fonction des incohérences causées par son application. L'identification des inconsistances est effectuée par le moteur Pellet et l'analyse du changement est basée sur la classification des incohérences détectées selon des patrons d'incohérences.
3. La *résolution des changements* qui permet de proposer des alternatives de résolution pour les inconsistances et permet de les évaluer afin de choisir celle qui préserve la consistance de l'ontologie évoluée.

La principale contribution de l'approche est d'intégrer une modélisation par patrons dans un processus de gestion des changements d'ontologies. Cependant, elle souffre de deux limites principales [Djedidi, 2009] : 1) des activités lourdes en terme de coût de traitement et de capacité mémoire (application temporaire de changements, classification et mise en correspondance des patrons, évaluation de l'impact des résolutions sur la qualité de l'ontologie évoluée, etc.) et 2) un prototype à fonctionnalités réduites nécessitant d'être complété pour permettre l'application de l'ensemble des patrons définis par l'approche.

Jaziri et al. [2010] ont défini une approche préventive de résolution d'inconsistances pour le suivi du processus d'évolution d'ontologies. Le travail traite deux types de cohérences : la cohérence inter version et la cohérence intra version. La cohérence inter version permet d'exprimer la validité de l'ontologie par rapport à ses différentes versions, alors que la cohérence intra version représente la cohérence interne propre à chaque version d'ontologie et indépendamment des autres. Pour assurer ces différentes cohérences, les auteurs proposent un ensemble de *kit de changements* regroupant les changements ontologiques et l'ensemble de leurs opérations correctrices. Ces kits sont formalisés par le langage formel Z [Spivey and Abrial, 1992] et implémentés en Java donnant naissance à l'outil *Consistology*. L'approche définit de nombreux types de changements simples et composés qui ont été appliqués sur une ontologie représentant le système d'éducation tunisien pour le faire évoluer vers le système L.M.D (Licence Master Doctorat).

Malgré la richesse des changements étudiés, le travail ne traite pas le peuplement d'ontologies : aucun changement n'est proposé pour décrire les individus et leurs assertions. De plus, l'approche se focalise sur les ontologies conceptuelles construites à partir d'un digramme de classes UML (Unified Modeling Language), ce qui a donné des règles de cohérence fortement liées aux contraires UML.

Gueffaz et al. [2012] ont proposé une méthodologie CLOcK (Change Log Ontology Checker) pour identifier les incohérences résultant du processus d'évolution d'ontologies. L'identification des inconsistances est assurée par le model cheking qui est une technique utilisant la logique temporelle [Pnueli, 1977] et un vérificateur de modèle, appelé "model checker", pour explorer tous les états possibles d'un système en vérifiant s'ils répondent à une propriété particulière [Clarke et al., 1999]. La méthodologie est décomposée en deux phases [Gueffaz, 2012] :

1. La *transformation de graphe sémantique en un langage de model checking* qui consiste à traduire le log d'évolution d'ontologie, spécifié en OWL DL, dans un formalisme compréhensible par le model checker. Deux outils ont été proposés pour réaliser cette transformation. En premier lieu, les auteurs ont développé l'outil RDF2SPIN qui transforme un

graphe RDF en langage PROMELA mais qu'il ne peut gérer que 255 états. Par ailleurs, la deuxième proposition était de développer RDF2NuSMV qui transforme un graphe RDF en langage NuSMV et qui est capable de gérer un plus grand nombre d'états.

2. *La vérification des propriétés et l'identification des inconsistances* qui permet de chercher les inconsistances à partir du graphe NuSMV présentant le fichier log de l'ontologie. Il s'agit, tout d'abord, de générer tous les motifs incohérents pouvant figurer dans la chronologie des axiomes du fichier log. Ensuite, le model checker NuSMV est utilisé pour vérifier si l'un des motifs défini en logique temporelle peut être localisé dans le graphe NuSMV.

Le travail propose le model checking comme un nouveau formalisme capable d'identifier les inconsistances représentées dans un log d'évolution d'ontologies. Cependant, aucune stratégie n'est présentée pour résoudre ces inconsistances.

Sellami et al. [2013] ont proposé un outil de construction et d'évolution d'ontologies à partir des textes, DYNAMO-MAS (DYNAMIC Ontology for information retrieval-Multi-Agent System). Le travail se place dans le cadre des RTO (Ressource Termino-Ontologique) qui permettent "*d'associer une partie terminologique et/ou linguistique aux ontologies afin d'établir une distinction claire entre la manifestation linguistique (le terme) et la notion qu'elle dénote (le concept)*" [Touhami et al., 2011]. L'approche proposée s'intéresse à la première phase du processus d'évolution d'ontologies qui est l'identification des changements [Stojanovic, 2004] et elle est décomposée en quatre phases [Sellami, 2012] :

1. *L'enrichissement du corpus* qui consiste à ajouter de nouveaux documents textuels au corpus.
2. *L'extraction de nouvelles connaissances* qui permet d'analyser les nouveaux documents pour identifier des nouvelles connaissances en sélectionnant les termes et les relations non représentés dans l'ontologie initiale.
3. *L'interprétation des connaissances identifiées et l'évolution de l'ontologie* qui permet d'intégrer les nouvelles connaissances à l'ontologie en proposant différentes alternatives d'intégration et d'évolution d'ontologie ;
4. *l'évaluation de suggestions d'évolution* qui consiste à faire intervenir l'ontologiste ou bien l'utilisateur pour évaluer la qualité des suggestions proposées par le système et choisir la plus appropriée.

Contrairement aux approches précédentes, l'approche DYNAMO-MAS présente une description détaillée et une implémentation de l'étape d'identification de besoins d'évolution d'un

domaine. Toutefois, elle ignore les étapes de formalisation des changements ontologiques et la résolution des inconsistances.

Pittet et al. [2014] ont proposé une approche de gestion des changements ontologiques appliquée sur les "Services Généraux" (en anglais "Facility Management") [Vanlande et al., 2008]. L'approche proposée est appelée OntoVersionGraph et se décompose en six phases :

1. La *capture des changements* qui permet d'identifier la liste des changements ontologiques faisant évoluer l'ontologie. Cette liste est établie par l'utilisateur.
2. La *modélisation des changements* qui permet de représenter les changements identifiés avec le langage OWL-DL.
3. La *sémantique des changements* qui identifie les inconsistances. Cette étape est réalisée en se basant sur les travaux de [Gueffaz et al., 2012].
4. L'*implémentation des changements* qui consiste à appliquer les changements ontologiques.
5. La *propagation des changements* qui permet de propager les changements vers les entités dépendantes.
6. La *validation des changements* qui consiste à valider les changements et les sauvegarder.

Le travail s'intéresse aux ontologies SHOIN(D) et adresse les changements d'ajout et de suppression de concepts, d'instances, de rôles et des data type [Pittet, 2014]. Pour résoudre les inconsistances, les auteurs ont appliqué une approche a posteriori basée sur le raisonneur Pellet.

Liu et al. [2014] ont introduit l'utilisation de SetPi-calcul [Milner, 1999] pour modéliser le processus d'évolution d'ontologies. Ils ont défini tout d'abord un nouveau formalisme pour représenter les ontologies par des entités de SetPi. Les classes, les propriétés et les instances sont modélisées comme des processus. Les relations de subsomption "is-a" sont modélisées par des "canaux". Les changements ontologiques sont considérés comme des messages échangés entre les processus.

Le travail considère de nombreux changements ontologiques (simples, composés et complexes). Cependant, il se focalise sur la formalisation et ne propose aucune implémentation concrète du modèle défini. En outre, il ne traite pas les incohérences.

Pour résumer, de nombreuses approches ont été proposées dans la littérature pour définir et implémenter le processus d'évolution d'ontologies. Le Tableau 2.4 présente certaines approches tout en précisant les langages utilisés, l'implémentation, la gestion des inconsistances et leurs spécificités. Ainsi, nous pouvons observer que différents langages ont été étudiés :

KAON [Stojanovic, 2004], RDF [Luong and Dieng-Kuntz, 2007], OWL [Klein, 2004, Djedidi and Aufaure, 2010], etc. En se basant sur ces langages, plusieurs changements ontologiques ont été définis et différentes classifications de ces changements ont été proposées [Stojanovic, 2004, Klein, 2004]. Ainsi, certains travaux se sont focalisés sur l'étude de l'enrichissement d'ontologies [Klein, 2004, Jaziri et al., 2010]. D'autres ont étudié aussi le peuplement d'ontologies [Luong and Dieng-Kuntz, 2007, Djedidi and Aufaure, 2010]. La résolution des inconsistances est encore insuffisamment étudié. En effet, certaines approches ont ignoré cet axe puisqu'ils se sont intéressés à d'autres problématiques, comme par exemple la gestion des versions des ontologies [Hartung et al., 2013]. D'autres travaux se sont focalisés plutôt sur l'identification des inconsistances sans les résoudre [Gueffaz et al., 2012]. Certains chercheurs se sont intéressés également par la résolution des inconsistances [Djedidi and Aufaure, 2010, Luong and Dieng-Kuntz, 2007, Pittet et al., 2014]. Ces approches admettent un processus a posteriori de traitement des inconsistances qui nécessite l'utilisation d'une ressource externe (tel qu'un raisonneur) afin de vérifier la consistance de l'ontologie évoluée. Le travail de Jaziri et al. [2010] présente une approche préventive de résolution d'inconsistances. Toutefois, les inconsistances étudiées sont limitées et les changements se focalisent sur le niveau structurel des ontologies et non sur la propagation de changement au niveau des individus et leurs assertions.

Approches	Spécification	Implémentation	Gestion des inconsistances	Spécificités
[<i>Stojanovic, 2004</i>]	KAON	Framework KAON	<ul style="list-style-type: none"> - Identification de certaines inconsistances. - Stratégies proposées à l'ontologiste pour résoudre les inconsistances. 	<ul style="list-style-type: none"> - Un processus global d'évolution d'ontologies. - Sauvegarde des versions de l'ontologie évoluée et traçabilité du processus d'évolution. - L'ensemble des contraintes de cohérence dépend fortement du langage KAON.
[<i>Klein, 2004</i>]	OWL	OntoView, PROMPTdiff	<ul style="list-style-type: none"> - Identification et résolution des inconsistances. 	<ul style="list-style-type: none"> - Une approche de gestion des changements pour les ontologies distribuées. - Identification de la différence entre les versions de l'ontologie évoluée. - Sauvegarde de la traçabilité des changements ontologiques.
[<i>Djedidi and Aufaure, 2010</i>]	OWL DL	Prototype Onto-EVO ⁴ L	<ul style="list-style-type: none"> - Identification des inconsistances en utilisant le raisonneur Pellet. 	<ul style="list-style-type: none"> - Approche basée sur les patrons de conception. - Évaluation de la qualité de l'ontologie évoluée. - Approche nécessitant des activités lourdes.
[<i>Jaziri et al., 2010</i>]	OWL	Prototype Consistology	<ul style="list-style-type: none"> - Identification et résolution des inconsistances 	<ul style="list-style-type: none"> - Approche basée sur des kits des changements englobants les changements ontologiques et leurs opérations correctrices. - Pas de propagation des changements pour les individus
[<i>Gueffaz et al., 2012</i>]	OWL DL	Prototype	<ul style="list-style-type: none"> - Identification des inconsistances en utilisant le checker NuSMV. 	<ul style="list-style-type: none"> - Approche d'évolution d'ontologies, CLOCK (Change Log Ontology Checker) basée sur le modèle checking. - Approche nécessitant la transformation des ontologies OWL au langage NuSMV.

[<i>Sellami et al.</i> , 2013]	OWL	DYNAMO-MAS	—	—	<ul style="list-style-type: none"> - Approche d'évolution des Ressources Terminologiques - Identification de besoins d'évolution
[<i>Hartung et al.</i> , 2013]	OBO (Open Biomedical Ontologies) et RDF	L'outil Conto-diff, L'application web OnEX	—	—	<ul style="list-style-type: none"> - Identification de la différence entre les versions de l'ontologie évoluée. - Approche basée sur le résultat d'un mapping semi-automatique calculé par des règles COG (Change Operation Generating).
[<i>Khattak et al.</i> , 2013b]	RDFS et OWL	Plugin Protégé	—	<ul style="list-style-type: none"> - Gérer les inconsistances en utilisant l'API KAON [<i>Stojanovic</i>, 2004]. 	<ul style="list-style-type: none"> - Approche de gestion des versions d'ontologies.
[<i>Liu et al.</i> , 2014]	OWL	—	—	—	<ul style="list-style-type: none"> - Proposition d'un nouveau formalisme, le SetPi-calcul, pour modéliser l'évolution d'ontologies. - Formalisation des changements élémentaires, composés et complexes.
[<i>Pittet et al.</i> , 2014]	SHOIN(D)	OntoVersioning API	—	<ul style="list-style-type: none"> - Identification des inconsistances en utilisant le raisonneur Pellet. 	<ul style="list-style-type: none"> - Approche d'évolution des ontologies SHOIN(D). - Évolution collaborative de l'ontologie et ses vues.

TABLE 2.4: Approches d'évolution d'ontologie.

2.5 Fusion d'ontologies

La fusion d'ontologies est "la création d'une nouvelle ontologie à partir de deux ou plusieurs ontologies existant avec des parties qui se chevauchent" [Klein, 2001]. La création de la *nouvelle ontologie* (aussi appelée l'ontologie globale) passe essentiellement par deux grandes phases : 1) l'alignement d'ontologies qui cherche les correspondances entre leurs concepts et 2) la fusion d'ontologies se basant sur l'alignement trouvé. Pour accomplir la première phase, plusieurs techniques de similarité ont été proposées dans la littérature. Elles sont généralement classées en quatre catégories : 1) les techniques lexicales, 2) les techniques structurelles, 3) les techniques sémantiques et 4) la combinaison des différentes techniques précédentes. Ainsi, les techniques lexicales considèrent les noms des entités et les comparent comme des chaînes de caractères, ex. la distance de Levenshtein [Levenshtein, 1966], distance de Stoilos [Stoilos et al., 2005], etc. Les techniques structurelles considèrent la structure des ontologies pour détecter les relations de subsomption [Batet et al., 2011], ex. Children et Leaves [Do and Rahm, 2002]. Quant aux techniques sémantiques, elles cherchent à identifier les relations sémantiques entre les concepts en utilisant des sources externes, ex. les ontologies linguistiques et les dictionnaires.

L'alignement d'ontologies est un axe de recherche largement étudié qui présente des résultats de plus en plus importants. Plusieurs états de l'art décrivant ces résultats sont disponibles dans la littérature, le lecteur pourra s'y référer [Euzenat and Shvaiko, 2013, Pavel and Euzenat, 2013, Dos Reis et al., 2015, Hamdi, 2011]. Dans cette section nous nous intéressons plutôt à la phase de fusion. Nous présentons ses besoins (pourquoi fusionner des ontologies ?), les stratégies de fusion (comment fusionner des ontologies ?) et les approches étudiant cette problématique (quels sont les travaux ayant contribué à la fusion d'ontologies ?).

2.5.1 Besoins de fusion

Les raisons expliquant le besoin de fusionner des ontologies peuvent être résumées par les deux points suivants :

- *Construction d'ontologies*. Plusieurs stratégies sont adoptées pour construire les ontologies : construction d'ontologies à partir de zéro [Gómez-Pérez et al., 1996], construction d'ontologies à partir des bases de données [Kamel and Aussenac-Gilles, Krivine et al., 2009, Cullot et al., 2007], construction d'ontologies à partir de textes [Maedche and Staab, 2000, Bourigault and Aussenac-Gilles, 2003, Zouaq and Nkambou, 2008], construction d'ontologies par briques [Royer et al., 2014] ou bien aussi construction d'ontologies à partir d'autres ontologies [Ushold and King, 1995]; ce qui se ramène à leur fusion.
- *Enrichissement d'ontologies*. Après leur construction et leur utilisation, les ontologies su-

bissent souvent des changements. Certaines adaptations nécessitent l'évolution des entités ontologiques (voir Section 2.4). D'autres adaptations demandent l'intégration de nouvelles ontologies, issues du même domaine ou de domaines connexes, pour enrichir l'ontologie initiale et ajouter des nouvelles connaissances.

2.5.2 Stratégies de fusion

Deux types d'approches de fusion d'ontologies sont distinguées dans la littérature : les approches symétriques (appelées en anglais "full merge") et les approches asymétriques (appelées "source-driven merging" et "target-driven merging").

Les approches symétriques permettent de fusionner des ontologies tout en préservant tous leurs éléments dans l'ontologie globale (O_G). L'idée est de conserver tous les composants des deux ontologies sans favoriser l'une d'elles. Ainsi, si on considère deux ontologies O_1 et O_2 , leur résultat de fusion donne : $O_G = Merge(O_1, O_2) = Merge(O_2, O_1)$. Ces approches doivent, ainsi, satisfaire les propriétés suivantes² [Raunich and Rahm, 2014] :

- (P1. *Préservation des entités*). Chaque entité (concept) des deux ontologies O_1 et O_2 doit avoir un élément correspondant dans l'ontologie globale.
- (P2. *Préservation des relations*). Chaque relation (is-a ou autre) doit être exprimée explicitement ou impliquée dans l'ontologie globale.
- (P3. *Préservation des instances*). Toutes les instances des deux ontologies doivent être préservées dans l'ontologie globale.
- (P4. *Préservation d'égalité*). Si deux concepts des deux ontologies sont égaux alors il n'y aura qu'un seul concept dans le résultat de fusion.

Les approches asymétriques distinguent deux types d'approches : "source-driven" et "target-driven". L'approche "source-driven" consiste à conserver les concepts de l'ontologie source dans l'ontologie globale et rajouter les concepts non redondants de l'ontologie cible. En revanche, pour l'approche "target-driven", ce sont les concepts de l'ontologie cible qui sont préservés. Ainsi, les approches asymétriques respectent les propriétés P3 et P4 alors que, pour le cas des approches "target-driven", P1 et P2 sont remplacées par les suivantes [Raunich and Rahm, 2014] :

- (P1'. *Préservation des éléments cibles*). Chaque élément de l'ontologie cible (target) est préservé dans l'ontologie globale.

2. Noter que ces propriétés ont été définies dans le cadre des taxonomies mais restent valables pour les ontologies.

- (P2'. *Préservation de relations cibles*). Chaque relation cible doit être exprimée explicitement ou implicitement dans l'ontologie globale.

2.5.3 Approches de fusion d'ontologies

Nous présentons dans cette section les principales approches de fusion d'ontologies en précisant leurs caractéristiques.

Stumme and Maedche [2001] ont proposé l'approche FCA-Merge qui permet de fusionner des ontologies en se basant sur l'analyse formelle de concepts, en anglais "Formal Concept Analysis" FCA [Ganter and Wille, 2012]. L'algorithme de fusion prend comme entrées deux ontologies et un ensemble de documents en langage naturel pour générer en sortie une nouvelle ontologie globale. Il est décomposé en trois phases :

1. analyse linguistique des deux ontologies en se basant sur l'ensemble de documents. Cette phase vise à déterminer les contextes des deux ontologies et extraire leurs instances ;
2. fusion des deux contextes et calcul de treillis [Birkhoff, 1967] en utilisant les techniques de FCA ;
3. génération de l'ontologie globale à partir du treillis construit. Cette phase est semi-automatique et se base sur l'intervention d'utilisateurs.

L'approche FCA-Merge est une approche symétrique. Elle se focalise sur les ontologies frame et se base sur le concept de treillis. L'algorithme de fusion ignore les conflits, c'est plutôt l'utilisateur qui se charge de les identifier et les corriger.

Noy and Musen [2003] ont proposé deux outils pour l'alignement et la fusion d'ontologies. L'outil AnchorPROMPT [Noy and Musen, 2001] cherche les points de similarité entre ontologies et IPROMPT [Noy and Musen, 2000] se charge de les fusionner. Le processus de fusion adopté est semi-automatique et se base sur l'intervention d'utilisateur. L'approche utilisée est principalement symétrique. Cependant, en cas de conflits, l'utilisateur intervient pour choisir la stratégie de résolution adéquate conduisant ainsi, à une solution partiellement asymétrique. L'algorithme peut être résumé par les étapes suivantes :

1. Identifier une première liste d'appariements des deux ontologies.
2. Une liste des opérations est suggérée à l'utilisateur (ex. fusionner des classes, fusionner des slots (les propriétés), etc.).
3. L'utilisateur peut choisir de déclencher l'une des opérations proposées par l'outil ou spécifier l'opération souhaitée directement.

4. L'outil effectue l'opération. Par exemple la fusion de deux classes est réalisée en suivant les étapes suivantes : 1) fusionner les slots liés aux classes ; 2) fusionner les instances ; 3) copier la classe de l'ontologie source à celle fusionnée ; 4) copier toute la profondeur de la classe ; 5) copier aussi tous les parents de la classe à la hiérarchie racine.
5. L'exécution des opérations peut engendrer des conflits. Une intervention de l'utilisateur est alors nécessaire pour les résoudre.

Les outils PROMPT sont disponibles en tant que plugins pour l'outil Protégé. Cependant, ils ne traitent que les ontologies Frame.

Kotis and Vouros [2004] ont proposé l'approche HCONE (Human-Centered Ontology Management Environment) permettant de fusionner deux ontologies d'une manière symétrique et semi-automatique. L'algorithme de fusion consiste, tout d'abord, à trouver l'ensemble de correspondances entre les ontologies en utilisant l'ontologie linguistique WordNet [Miller, 1995] et l'analyse sémantique latente (en anglais "Latent Semantics Analysis", LSA) [Dumais, 2004]. Ensuite, il fusionne les ontologies à l'aide de l'intervention d'utilisateur, notamment pour éviter les inconsistances. Les auteurs se focalisent sur la phase d'alignement et ne fournissent pas de détails sur la phase de fusion.

Li et al. [2010] proposent le framework MMOMS (Multi-Mapping based Ontology Merging System) pour fusionner des ontologies OWL. MMOMS est composé de plusieurs modules assurant :

- la recherche de similarité entre ontologies en se basant sur des techniques d'apprentissage automatique, en anglais "machine learning" [Goldberg and Holland, 1988], l'ontologie linguistique WordNet et des techniques structurelles ;
- la fusion d'ontologies en se basant sur les similarités identifiées. L'algorithme de fusion traite les concepts, les relations et les attributs des deux ontologies ;
- le traitement de certains conflits sémantiques et structurels.

Raunich and Rahm [2014] ont proposé une approche de fusion des taxonomies qui se compose de deux phases : préliminaire et principale.

- La phase préliminaire prend deux taxonomies et leur résultat de correspondance pour générer un "graphe intégré". L'identification des correspondances est basée sur les travaux de Chiticariu et al. [2008] et l'outil COMA³ développé au sein de la même équipe de recherche Do and Rahm [2002].

3. <http://dbs.uni-leipzig.de/Research/coma.html>

- La phase principale assure la fusion des deux taxonomies source et destination. Elle adopte pour ceci une approche asymétrique appelée ATOM (Automatic Target-driven Ontology Merging) qui est de même le nom de prototype développé.

L'outil ATOM prend en entrée des taxonomies aux formats OWL, XSD ou bien XML et génère d'une manière automatique le résultat de leur fusion. Le mapping entre les deux taxonomies peut être identifié manuellement (par un expert de domaine) ou bien d'une manière semi-automatique (grâce à l'utilisation d'un ensemble d'outils). Les auteurs proposent une approche capable d'éviter la redondance et de réduire un ensemble de conflits dans l'ontologie résultat. Cependant, ce travail se focalise sur les taxonomies et ne traite que les ontologies composées par des concepts et des relations hiérarchiques (is-a).

Pour résumer, peu d'approches ont été proposées dans la littérature pour définir et implémenter le processus de fusion d'ontologies. Le Tableau 2.5 synthétise les approches présentés ci-dessus en précisant leurs types (symétrique ou bien asymétrique), les langages utilisés, l'implémentation et la gestion des inconsistances.

Approche	Type	Spécification	Implémentation	Spécificités & Gestion des conflits
[Stumme and Maedche, 2001]	Symétrique	Frame	FCA-Merge	- Approche semi automatique - Approche basée sur l'analyse formelle de concepts - Pas de gestion de conflits
[Noy and Musen, 2000]	Symétrique & Asymétrique	Frame	Prompt	- Approche semi automatique - Détection des conflits - Intervention d'utilisateur pour la résolution des conflits
[Kotis and Vouros, 2004]	Symétrique		HCONE	- Approche semi automatique - Approche basée sur les treillis et l'analyse sémantique latente - Pas de gestion de conflits
[Li et al., 2010]	Symétrique	OWL	MMOMS	- Approche automatique - Gestion de certains conflits sémantiques et structurels
[Raunich and Rahm, 2014]	Asymétrique	Taxonomie en OWL	ATOM	- Approche automatique - Suppression de redondances - Gestion de certains conflits structurels

TABLE 2.5: Approches de fusion d'ontologies.

2.6 Bilan et positionnement

Dans ce chapitre, nous avons présenté les ontologies, leurs composants, leurs langages de représentation et leur cycle de vie. Nous avons également exposé les besoins et les enjeux de leur réutilisation, notamment pour les processus d'évolution et de fusion. Un état de l'art a été dressé pour synthétiser les travaux traitant les problématiques d'évolution et de fusion d'ontologies. D'après cet état de l'art, nous avons pu constater que pour l'évolution, la préservation de la consistance de l'ontologie évoluée et la résolution des incohérences sont encore des problématiques insuffisamment étudiées. De plus, les approches qui étudient les inconsistances, les résolvent d'une manière a posteriori en se basant sur des ressources externes (des raisonneurs ou/et des experts). Le Tableau 2.6 présente une classification des approches traitant l'évolution d'ontologie et précise notre positionnement par rapport à elles. Ainsi, nous nous sommes intéressés à proposer une approche a priori de résolution d'inconsistances permettant d'éviter les retours en arrière après l'application des changements. Nous étudions les ontologies OWL et nous traitons à la fois leur niveau structurel et assertionnel. Plusieurs changements (élémentaires, composés et complexes) sont étudiés pour répondre aux besoins d'évolution les plus larges possible [Mahfoudh et al., 2015a].

Pour la fusion d'ontologies, nous proposons une approche automatique traitant les ontologies OWL qui est capable d'éviter la redondance de données et de réduire les conflits dans l'ontologie globale [Mahfoudh et al., 2014c]. Le Tableau 2.7 présente notre positionnement par rapport aux approches existantes.

Dans le but d'atteindre les objectifs visés, nous proposons d'utiliser le formalisme mathématique des grammaire des graphes présenté dans le chapitre suivant.

Approches	Niveau		Changements		Inconsistances			
	Structure	Assertif	Élémentaires	Composées et complexes	Sans traitement d'inconsistances	Identification des inconsistances	Résolution a postériori	Résolution a priori
[<i>Klein and Noy, 2003</i>]	✓		✓	✓			✓	
[<i>Stojanovic, 2004</i>]	✓	✓	✓	✓				✓
[<i>Luong and Dieng-Kuntz, 2007</i>]	✓	✓	✓	✓			✓	
[<i>Djedidi and Aoufaure, 2010</i>]	✓	✓	✓	✓			✓	
[<i>Jaziri et al., 2010</i>]	✓		✓	✓				✓
[<i>Gueffaz et al., 2012</i>]	✓	✓	✓			✓		
[<i>Sellami et al., 2013</i>]	✓		✓		✓			
[<i>Hartung et al., 2013</i>]	✓	✓	✓	✓	✓			
[<i>Pittet et al., 2014</i>]	✓	✓	✓	✓			✓	
[<i>Liu et al., 2014</i>]	✓		✓	✓	✓			
[<i>Mahfouh et al., 2015a</i>]	✓	✓	✓	✓				✓

TABLE 2.6: Positionnement et classification des approches d'évolution d'ontologies.

Approches	Type d'ontologies			Symétrie		Conflits		
	Frame	Taxonomie	OWL	Symétrique	Asymétrique	Sans gestion de conflits	Résolution manuelle	Résolution automatique
[Stumme and Maedche, 2001]	✓			✓		✓		
[Noy and Musen, 2000]	✓			✓			✓	
[Kotis and Vourros, 2004]				✓		✓		
[Li et al., 2010]			✓	✓				✓
[Raurich and Rahm, 2014]		✓			✓			✓
[Mahfouh et al., 2014c]			✓		✓			✓

TABLE 2.7: Positionnement et classification des approches de fusion d'ontologies.

Chapitre 3

Grammaires de graphes

Sommaire

3.1 Introduction	61
3.2 Graphes et Grammaires de Graphes	62
3.2.1 Préliminaires	62
3.2.2 Grammaires de Graphes	64
3.3 Approches algébriques de transformations de graphes	68
3.3.1 Théorie des Catégories	68
3.3.2 Approche Double PushOut (DPO)	70
3.3.3 Approche Simple PushOut (SPO)	71
3.3.4 Exemple d'application du SPO sur les GGT	72
3.4 Outils de transformation de graphes	74
3.4.1 FUJABA	74
3.4.2 VIATRA	74
3.4.3 ATOM3	74
3.4.4 AGG	75
3.5 Applications des grammaires de graphes aux ontologies	75
3.5.1 Travaux liés à l'interrogation d'ontologies	76
3.5.2 Travaux liés à l'évolution d'ontologies	76
3.5.3 Travaux liés à la fusion d'ontologies	77
3.6 Bilan	78

"Les mathématiques sont le langage de l'Univers."

Galileo Galilei

3.1 Introduction

Les graphes règnent en maître dans notre vie. Ils sont utilisés partout : cartes routières, cartes cognitives, réseaux sociaux, arbres généalogiques, jeux, etc. Ils sont ainsi nommés parce

qu'ils peuvent être représentés graphiquement et c'est cette représentation graphique qui nous aide à comprendre leurs propriétés et les relations entre les objets représentés. Cela dit, derrière cette représentation "simple" de sommets et d'arêtes, se trouvent des formalismes mathématiques capables de spécifier des problèmes complexes.

Partant de ce principe et sachant que les ontologies peuvent aussi être représentées sous forme de graphes (voir Chapitre 2, Section 2.2), nous avons pensé à l'utilisation de ce formalisme dans notre travail. Nous nous sommes intéressés, plus particulièrement, aux grammaires de graphes qui permettent aussi bien de représenter des graphes que de spécifier leur évolution.

Les grammaires de graphes, connues aussi sous le nom de la réécriture de graphes, sont utilisées dans plusieurs domaines de l'informatique [Ehrig *et al.*, 1996] tel que le génie logiciel [Schneider, 1997, da Costa and Ribeiro, 2012, Khalifaoui *et al.*, 2013], la reconnaissance de formes [Flasiński and Jurek, 2014], l'optimisation [Assmann, 1996], l'imagerie [Lladós and Sánchez, 2003], etc. Récemment, elles ont commencé à être utilisées dans le domaine des ontologies, par exemple dans la formalisation des ontologies modulaires [d'Aquin *et al.*, 2007], l'interrogation des graphes RDF [Braatz and Brandt, 2008] et la journalisation des changements ontologiques [Javed *et al.*, 2013].

L'idée principale des grammaires de graphes consiste à modifier des graphes avec des règles de réécriture. Pour ceci, et selon le problème spécifié, elles se basent sur une des approches de transformation de graphes permettant de transformer un graphe d'un état à un autre tout en respectant un ensemble de contraintes.

Ainsi, l'objectif de ce chapitre est d'introduire les grammaires de graphes et les transformations de graphes. Nous présentons en premier lieu les concepts nécessaires à la compréhension du reste du manuscrit. Nous abordons ensuite les approches algébriques de transformation de graphes et les principaux outils permettant de les implémenter. Enfin, nous présentons les principaux travaux qui se sont intéressés à l'application des grammaires de graphes dans le domaine des ontologies.

3.2 Graphes et Grammaires de Graphes

3.2.1 Préliminaires

Nous présentons dans cette section les définitions des concepts fondamentaux de graphes qui seront utilisés dans ce manuscrit.

Définition 1 (Graphe Orienté). Un graphe orienté $G(N, E)$ est une structure composée par un ensemble de nœuds (N), d'arêtes (E) et d'une application $s : E \rightarrow N \times N$ qui à chaque arête associe un nœud source et cible (Figure 3.1.(a)). À Noter qu'un graphe est dit non orienté si $\forall e, s(e) = (x, y) \Rightarrow \exists e', s(e') = (y, x)$.

Définition 2 (Graphe étiqueté). Un graphe étiqueté est un graphe (orienté ou non orienté) auquel on associe à ses arêtes un ensemble non vide d'étiquettes (labels) $e : E \rightarrow L$ avec L est un ensemble de labels (Figure 3.1.(b)).

Définition 3 (Graphe attribué). Un graphe attribué est un graphe étendu par un ensemble d'attributs A , une fonction d'attribution $att : N \cup E \rightarrow \mathcal{P}(A)$, \mathcal{P} représentant l'ensemble des parties, et une fonction d'évaluation $val : A \rightarrow V$ (Figure 3.1.(c)). Ainsi, chaque nœud ou arête peut avoir un ensemble d'attributs $\mathcal{P}(A)$ dont les valeurs sont données par val .

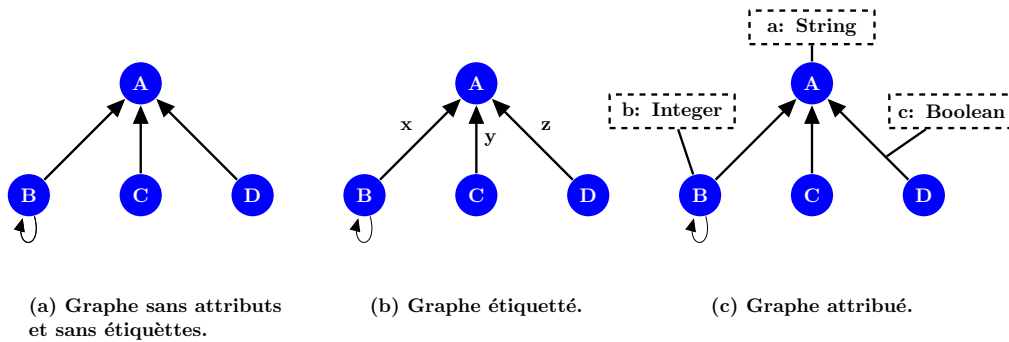


FIGURE 3.1 – Exemples des types de graphes.

Définition 4 (Sous-graphe). Un graphe G' inclus dans un autre graphe G ($G' \subseteq G$) est dit un sous-graphe de G .

Définition 5 (Pattern de graphe). Un pattern de graphe (en anglais "Graph Pattern") est un motif $PG(N \cup X_N, E \cup X_E)$ avec des variables (X) pouvant substituer des nœuds et arêtes d'un graphe.

Définition 6 (Substitution). La substitution est une opération qui consiste à remplacer un sous graphe d'un graphe par un autre sous graphe.

Définition 7 (Morphisme de graphes). Soient deux graphes $G(N, E)$ et $G'(N', E')$, un morphisme de graphes $m(f, g)$ est une application de G à G' définie par deux applications $f : N \rightarrow N'$ et $g : E \rightarrow E'$ (Figure 3.2). Un morphisme doit préserver la structure, c.à.d si $e = (s, t)$ et $g(e) = e' = (s', t')$ alors $s' = f(s)$ et $t' = f(t)$. En particulier, la recherche d'un pattern dans un graphe reviendra à trouver un morphisme.

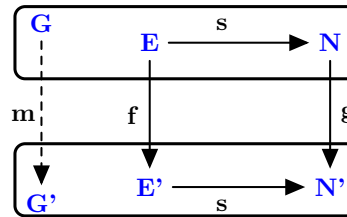


FIGURE 3.2 – Morphisme de graphes.

Définition 8 (Typage). Le typage est un morphisme d'un graphe $G(N, E)$ à un graphe type $GT(N_T, E_T)$ avec N_T correspond aux types des nœuds et E_T aux types des arêtes.

3.2.2 Grammaires de Graphes

Rappelons que notre étude sur les graphes avait pour but d'identifier un formalisme capable de gérer les changements d'ontologies et répondant au mieux aux enjeux de leur adaptation. Ceci nous a amené à étudier les grammaires de graphes qui sont un formalisme de représentation et de spécification de transformations de graphes. Les grammaires de graphes font partie de la famille des systèmes de réécriture [Dershowitz and Jouannaud, 1989] qui ont pour but de transformer des objets (mots, termes, graphes, etc.) en appliquant des règles de réécriture. La réécriture a été initialement définie sur les mots (des chaînes de caractères) et s'exprime par les grammaires (un formalisme permettant de définir un langage formel à partir d'un alphabet donné [Hopcroft and Ullman, 1969]). Ce modèle de calcul a été repris et adapté pour réécrire les graphes, ce qui a donné naissance au formalisme de grammaires de graphes.

Formellement, une grammaire de graphe (GG) est une structure mathématique définie par $GG = (G, R)$ avec :

- $G(N, E)$ un graphe initial, appelé aussi graphe hôte ;
- R un ensemble de règles de réécriture, connues aussi sous le nom de règles de production.

Elles permettent de transformer un graphe hôte G et se sont définies par une paire de

graphes pattern $R(LHS, RHS)$ avec :

1. *LHS* (Left Hand Side) représente la pré-condition de la règle de réécriture et décrit la structure qu'il faut trouver dans un graphe hôte G pour pouvoir appliquer la règle.
2. *RHS* (Right Hand Side) représente la post-condition de la règle de réécriture et doit remplacer *LHS* dans G .

Les règles de réécriture peuvent également avoir des conditions supplémentaires appelées *NAC* (Negative Application Conditions). Ce sont des graphes pattern définissant des conditions qui ne doivent pas être vérifiées pour que la règle de réécriture puisse être appliquée.

Les graphes peuvent être typés, ce qui nous amène à parler des grammaires de graphes typés (*GGT*). Ces dernières sont souvent utilisées pour gérer des problèmes de modélisation nécessitant de spécifier et de préciser le type des objets modélisés. Ainsi, une grammaire de graphe typé est une structure définie par $GGT = (GT, GG) = (GT, G, R)$ avec :

- $GT(N_T, E_T)$ est un graphe type précisant le type des nœuds et des arêtes de graphe hôte G . Le typage de G est donné ainsi par un morphisme $t : G \rightarrow GT$ avec $t : E \rightarrow E_T$ et $t : N \rightarrow N_T$.

Principe de transformation de graphes

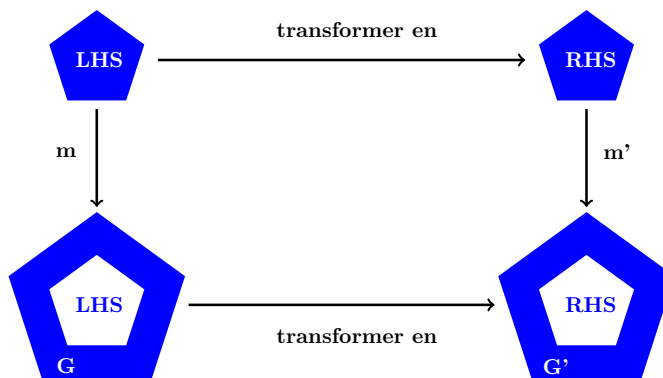


FIGURE 3.3 – Principe de transformation de graphes.

Le principe général de transformation de graphes est présenté par la Figure 3.3. Il s'agit d'appliquer une règle de réécriture $(LHS, RHS) \in R$, sur un graphe hôte G pour le transformer en un nouveau graphe G' . Afin d'appliquer la transformation, il faut chercher en premier lieu,

une occurrence du graphe LHS dans le graphe G . Cette identification est réalisée par un morphisme $m : LHS \rightarrow G$. Ensuite, il faut remplacer LHS par le graphe RHS qui doit figurer, par conséquent, dans le nouveau graphe G' , avec $m' : RHS \rightarrow G'$.

Le remplacement (transformation) de LHS par RHS peut être réalisé selon deux types d'approches [Rozenberg, 1999] : les approches ensemblistes (essentiellement les approches "Node Replacement" et "Edge Replacement") et les approches algébriques.

Approche "Node Replacement"

L'approche "Node Replacement" a été proposée par Engelfriet and Rozenberg [1991]. Son idée principale consiste à substituer un seul nœud (le LHS) par un nouveau sous-graphe (le RHS). Différentes variantes ont été proposées pour cette famille d'approches. La Figure 3.4 présente un exemple de transformation selon l'approche "Node Label Controlled" [Engelfriet and Rozenberg, 1991]. Celle-ci se base sur : (1) les étiquettes des nœuds et (2) les instructions de connexions qui permettent de rattacher le graphe RHS au graphe G . Dans l'exemple illustré, les instructions de connexion consistent à ajouter les arêtes (A,C), (B,B) et (X,B). Le graphe pattern LHS de la règle de réécriture est composé d'un seul nœud étiqueté "X". La première étape de la transformation consiste à trouver tous les nœuds étiquetés "X" dans le graphe G . Ensuite, il faut les remplacer par le graphe RHS . Les arêtes liant le nœud "X" à G seront supprimés. De nouvelles arêtes seront donc ajoutées à G' pour rattacher le graphe RHS au graphe initial et ceci selon le principe suivant : créer une arête entre chaque nœud du graphe RHS et les nœuds voisins du nœud "X" dans G en se basant sur les instructions de connexion.

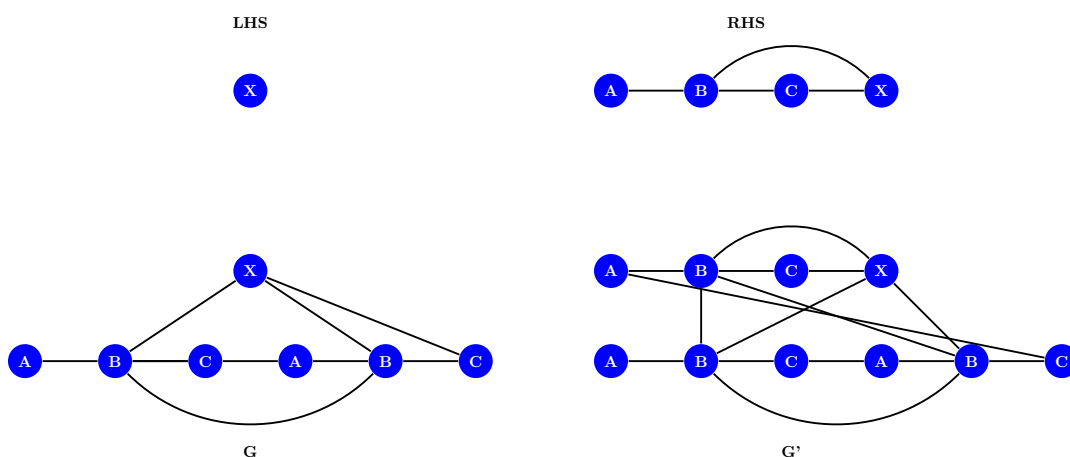


FIGURE 3.4 – Transformation de graphes selon l'approche "Node Label Controlled".

Approche "Edge Replacement"

L'approche "Edge Replacement" a été proposée par *Drewes et al.* [1999]. Elle consiste à remplacer une seule arête d'un graphe initial G par un graphe RHS pour obtenir un nouveau graphe G' . La Figure 3.5 montre un exemple d'application de cette approche. La transformation commence par identifier l'arête "e" pour le supprimer du Graphe G . Ensuite, il faut fusionner le nœud initial "début" du graphe RHS avec la source de l'arête "e" et le nœud fin "fin" du graphe RHS avec la cible de l'arête "e".

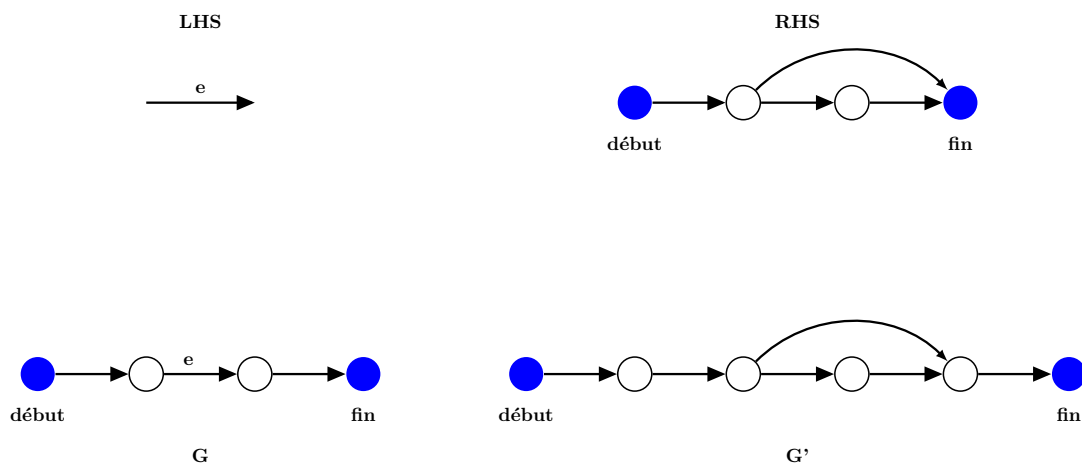


FIGURE 3.5 – Transformation de graphes selon l'approche "Edge Replacement".

Discussion

Les deux approches "Node Replacement" et "Edge Replacement" permettent de spécifier plusieurs problèmes de modélisation. Cependant, elles souffrent de deux limites principales :

- *Ignorance du contexte* : les transformations ne peuvent s'appliquer que sur une seule entité atomique, un nœud dans le cas de l'approche "Node Replacement" ou bien un arête pour l'approche "Edge Replacement". Ceci restreint le domaine d'application de ces deux approches. En particulier, pour les ontologies aucun changement composite ou complexe ne peut être envisagé.
- *Incapacité de gérer les attributs* : les deux approches "Node Replacement" et "Edge Replacement" ne peuvent pas gérer les graphes attribués. Ceci fait qu'elles ne peuvent être utilisées dans les problématiques de transformations de modèles qui ont souvent besoin

de représenter des attributs [Rebout, 2008].

Afin de remédier à ces limites, les approches algébriques se présentent comme une alternative rigoureuse et intéressante, capables à la fois de modéliser les transformations des graphes attribués et de définir des transformations complexes en tenant compte de plusieurs éléments (nœuds et arêtes) des graphes.

3.3 Approches algébriques de transformations de graphes

Deux variantes d'approches algébriques sont proposées pour les transformations de graphes : le *Double PushOut DPO* [Ehrig, 1979] et le *Simple PushOut SPO* [Löwe, 1993]. Cette famille d'approches se basent sur le concept "pushout" issu de la théorie des catégories. Ainsi, avant de les introduire et présenter une comparaison entre elles, une brève présentation des concepts de base de la théorie des catégories est indispensable.

3.3.1 Théorie des Catégories

La théorie des catégories est une branche de mathématiques qui étudie les relations entre des structures composées par des objets et des flèches (morphismes) entre ces objets [Barr and Wells, 1990]. Plus précisément, une catégorie \mathcal{C} est une structure composée de :

1. une collection d'objets O ;
2. un ensemble de morphismes M et une fonction $s : M \rightarrow O \times O$, $s(f) = (A, B)$ est notée alors $f : A \rightarrow B$;
3. une loi de composition associative $(\circ) : M \times M \rightarrow M$;
4. un morphisme identité pour chaque objet $id : O \rightarrow O$. La loi de composition doit avoir l'identité comme élément neutre.

Exemples des catégories utilisées dans la littérature :

- *Set* est la catégorie des ensembles. Ses objets sont les ensembles et ses morphismes sont les fonctions d'un ensemble. Elle est notée également *Ens*.
- *Graph* est la catégorie des graphes. Ses objets sont des graphes et ses morphismes sont appelés morphismes de graphes. Dans notre travail, nous nous intéressons en particulier à cette catégorie.

La théorie des catégories définit plusieurs concepts intéressants. Dans ce qui suit, nous présentons ceux qui nous serviront dans notre travail.

Span. Le span est un diagramme de la forme $O_1 \leftarrow O \rightarrow O_2$, avec O, O_1 et O_2 sont trois objets d'une catégorie \mathcal{C} et deux morphismes $m_1 : O \rightarrow O_1$ et $m_2 : O \rightarrow O_2$. À noter que le concept dual (l'opérateur inverse) de span est appelé le cospan.



FIGURE 3.6 – Les concepts span et cospan de la théorie des catégories.

Pushout. Soient trois objets O_1, O_2 et O_3 et deux morphismes $m_1 : O_1 \rightarrow O_2$ et $m_2 : O_1 \rightarrow O_3$. Le pushout de O_2 et O_3 consiste à :

1. un objet P et deux morphismes $m'_1 : O_2 \rightarrow P$ et $m'_2 : O_3 \rightarrow P$ avec $m'_1 \circ m_1 = m'_2 \circ m_2$;
2. pour tout morphisme $m''_1 : O_2 \rightarrow P'$ et $m''_2 : O_3 \rightarrow P'$ tel que $m_1 \circ m''_1 = m_2 \circ m''_2$, il y a un seul morphisme $m_3 : P \rightarrow P'$ tel que $m'_1 \circ m_3 = m''_1$ and $m'_2 \circ m_3 = m''_2$.

À noter que la forme duale de pushout est appelée le pullback.

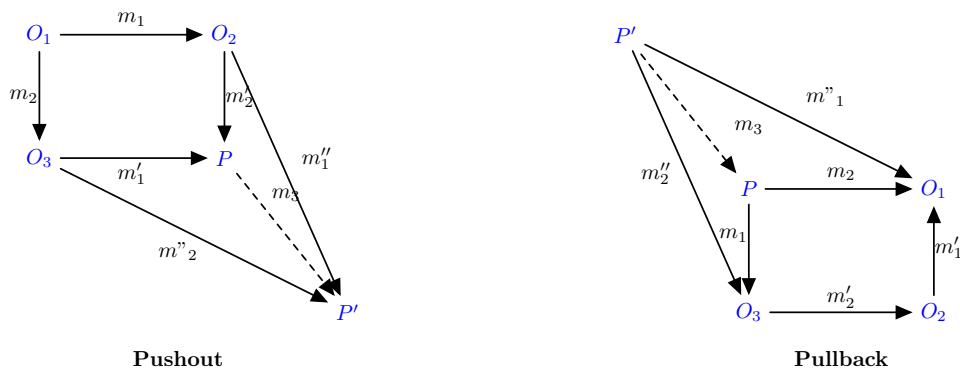


FIGURE 3.7 – Les concepts pushout et pullback de la théorie des catégories.

3.3.2 Approche Double PushOut (DPO)

L'approche Double Pushout (DPO) a été proposée par *Ehrig* [1979]. Elle se base en particulier sur la catégorie *Graph* et consiste à appliquer deux pushouts consécutifs afin de transformer un graphe G en un nouveau graphe G' . Les règles de réécriture sont sous la forme $(LHS \leftarrow I \rightarrow RHS)$ où I est un graphe, appelé graphe d'interface ou invariant. Ce graphe présente la structure commune entre LHS et RHS avec $I = (LHS \cap RHS)$.

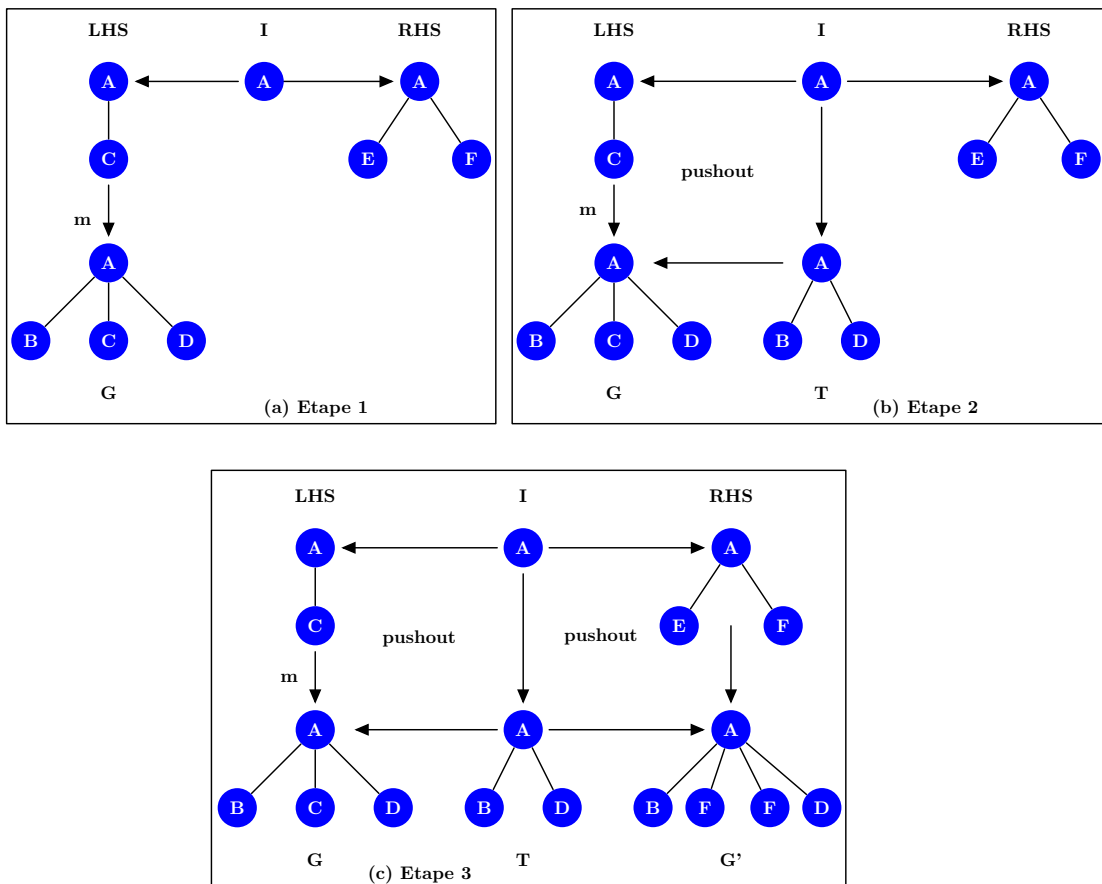


FIGURE 3.8 – Les étapes de l'application de l'approche DPO.

Les étapes d'application de l'approche DPO sont illustrées par la Figure 3.8 et consistent à :

1. trouver l'occurrence de LHS dans G à l'aide d'un morphisme $m : LHS \rightarrow G$.
2. supprimer de G le sous graphe $m(LHS) - m(I)$. Cette transformation est appliquée par un premier pushout et retourne un graphe temporaire T .

3. ajouter au graphe T , le sous graphe $m(RHS) - m(I)$. Cette étape est réalisée par un deuxième pushout et produit le nouveau graphe G' .

L'application de l'approche DPO impose la satisfaction de la *condition de collage*, appelée en anglais *Gluing Condition*, et qui consiste en deux conditions [Heckel et al., 2002] :

- la *condition de suspension*, appelée en anglais "*Dangling Condition*". Cette condition exige qu'une règle de réécriture ne peut être appliquée que si son application ne va pas causer l'apparition d'arêtes suspendues (c'est-à-dire, des arêtes qui ont une extrémité non liée à un nœud).
- la *condition d'identification* assure que chaque élément du graphe hôte G pouvant être supprimé suite à l'application de la règle de réécriture doit avoir une seule occurrence dans le graphe LHS .

3.3.3 Approche Simple PushOut (SPO)

L'approche Simple PushOut (SPO) est une approche algébrique de transformation de graphes proposée par Löwe [1993]. Sa différence majeure avec l'approche DPO est qu'elle ignore l'application de la condition de collage. Ceci lui permet d'appliquer des transformations que le DPO interdit.

Les transformations avec l'approche SPO sont assurées par l'application d'un seul pushout, à la différence de l'approche DPO qui nécessite deux pushouts. Les étapes de transformations sont les suivantes :

1. identifier le graphe LHS dans G selon un morphisme $m : LHS \rightarrow G$;
2. supprimer du graphe G , le graphe $m(LHS) - m(LHS \cap RHS)$ et supprimer tous les arêtes suspendues ;
3. ajouter le graphe $m(RHS) - m(LHS \cap RHS)$ au graphe hôte G .

La Figure 3.9 représente un exemple d'application d'une règle de réécriture selon l'approche SPO. Cette règle n'aurait pas été exécutée si elle était définie par l'approche DPO. En effet, nous pouvons constater que la deuxième étape de transformation qui consiste à supprimer le graphe $m(LHS) - m(LHS \cap RHS)$ du graphe G donne une arête suspendue, celui qui a comme extrémité le nœud "D".

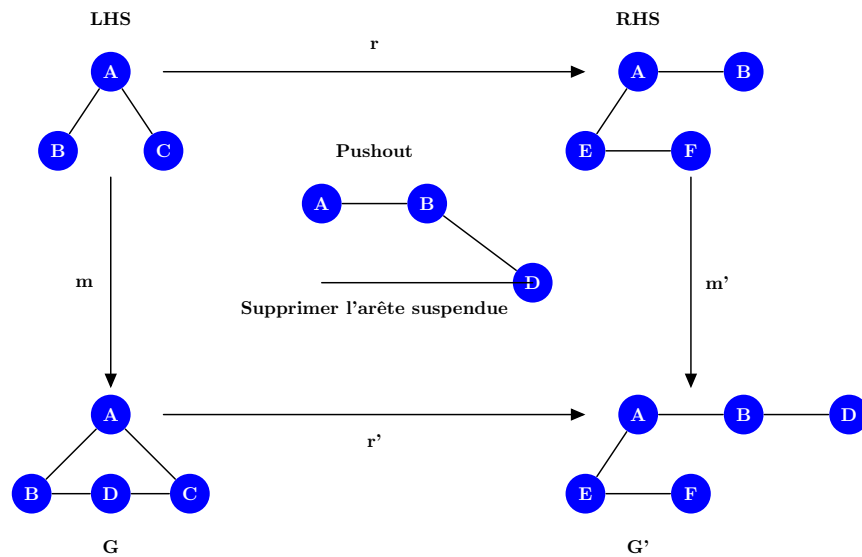


FIGURE 3.9 – Application d’une règle de réécriture selon L’approche SPO.

Discussion

La proposition de l’approche SPO avait pour objectif de remédier aux limites de l’approche DPO et d’enrichir ainsi le champ d’application des approches algébriques. En effet, en ignorant la condition de collage, l’approche SPO permet d’appliquer une large variété de transformations et permet de spécifier des problèmes ne pouvant pas être gérés par l’approche DPO. En particulier, dans le domaine des ontologies, l’approche DPO ne serait pas capable de définir tous les types de changements ontologiques (voir Chapitre 4).

3.3.4 Exemple d’application du SPO sur les GGT

Pour illustrer cette section, nous présentons un exemple simple et concret de l’utilisation des transformations algébriques et des grammaires de graphes. Nous choisissons de spécifier le jeu de Dames avec les grammaires de graphes typés (GGT) et l’approche algébrique SPO. Ainsi, la première étape consiste à construire le graphe type qui va présenter les éléments du jeu, à savoir le pion, les cases, le damier, etc. et les différentes relations entre eux. La Figure 3.10 présente le graphe type, le graphe hôte (le damier) et le morphisme de typage entre eux. Après avoir construit le graphe hôte, nous définissons notre règle de réécriture. Nous prenons comme exemple, la règle qui consiste à déplacer un pion noir de la case "B6" à une autre case "C5" (voir Figure 3.11). La transformation est réalisée par l’approche SPO et permet de vérifier si toutes les conditions sont satisfaites pour appliquer le changement. En particulier, les NAC

(NAC1 et NAC2) permettent d'interdire le déplacement du pion si la case "C5" est déjà occupée par un autre pion.

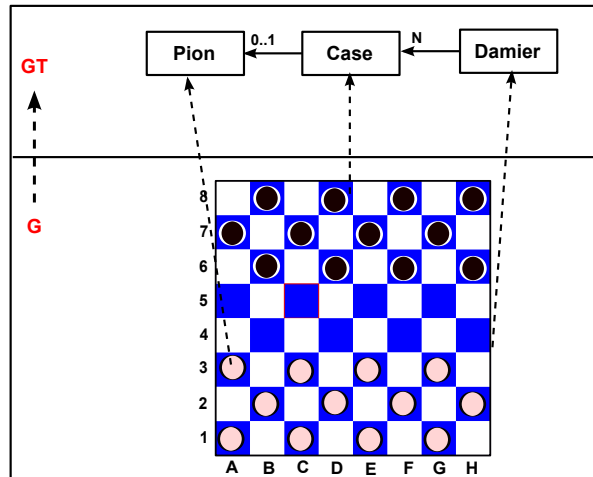


FIGURE 3.10 – Relation de typage avec le jeu de dame.

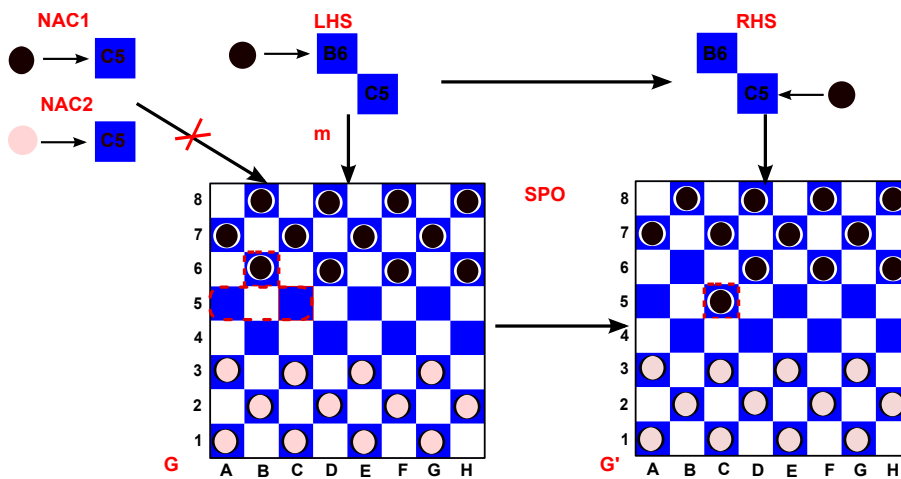


FIGURE 3.11 – Exemple d'application d'une règle de réécriture avec l'approche SPO.

3.4 Outils de transformation de graphes

Plusieurs outils ont été proposés pour mettre en œuvre les transformations de graphes comme par exemple AGG [Ermel. et al., 1999], Fujaba [Nickel et al., 2000], Viatra [Varró and Pataricza, 2004], ATOM3 [De Lara and Vangheluwe, 2002], GReAT [Balasubramanian et al., 2006], PROGRES [Schürr et al., 1999], etc. Une description détaillée de ces outils est disponibles dans la littérature, le lecteur pourra s’y référer [Czarnecki and Helsen, 2006, Mens et al., 2006]. Dans cette section nous présentons une brève description des outils les plus importants.

3.4.1 FUJABA

FUJABA¹ (From UML to Java and back again) est un outil principalement utilisé pour la génération de code Java à partir de diagrammes UML [Nickel et al., 2000]. Il utilise UML comme langage de modélisation visuel et exige ainsi que les graphes transformés soient représentés par les diagrammes UML. FUJABA est très répandu dans le domaine de transformations de modèle. Il supporte également les grammaires de graphes en particulier les techniques des grammaires triples, appelée en anglais Triple Graph Grammars, [Schürr, 1995] et qui sont une extension des grammaires de graphes. À noter que Fujaba ne supporte pas les approches algébriques de transformations de graphes.

3.4.2 VIATRA

VIATRA² (Visual Automated model TRAnsformations) est un plugin Eclipse de transformation de graphes [Varró and Pataricza, 2004]. Il utilise VPM (Visual and Precise Metamodeling) [Varró and Pataricza, 2003] comme langage de modélisation visuel. Il est principalement conçu pour supporter les transformations de modèles et les techniques de MDA (Model Driven Application Development). Il est aussi capable de gérer les grammaires de graphes. L’outil VIATRA propose l’utilisation des motifs de négations (l’équivalent des conditions d’application négatives) et offre la possibilité d’ordonner l’application des règles de réécriture.

3.4.3 ATOM3

L’outil ATOM3³ (A Tool for Multi-formalism and Meta-Modelling) a été proposé par De Lara and Vangheluwe [2002]. Il utilise le formalisme Entité-Relation (ER) ou les diagrammes de classes UML comme langage de modélisation visuelle. ATOM3 permet la transformation des

1. www.fujaba.de
2. eclipse.org/viatra
3. atom3.cs.mcgill.ca

graphes et offre la possibilité aux utilisateurs de gérer l'ordre d'application des règles de réécritures. Il permet également, et d'ailleurs c'est une caractéristique qui le distingue, de spécifier les contraintes par le langage OCL (Object Constraint Language).

3.4.4 AGG

AGG (Attributed Graph Grammar) est un outil de transformation des graphes typés et attribués [Ermel. et al., 1999]. Il est considéré comme l'un des outils d'usage général pour la transformation des graphes attribués et il est l'outil le plus répandu et cité dans le domaine. AGG a été développé dans le but d'implémenter les approches algébriques SPO et DPO. Il offre un cadre visuel permettant de définir les règles de réécriture d'une manière graphique et simple. Il permet également de définir des stratégies d'application de ces règles avec un mécanisme de niveaux de priorités (layers). AGG permet de sauvegarder le graphe type, les graphes hôtes et les règles de réécriture dans un même fichier de format GGX. Il est important de noter qu'AGG offre une API Java⁴ permettant son intégration dans des applications Java.

Discussion

Les grammaires de graphes se distinguent par la richesse des outils qu'elles fournissent. Selon la nature de problème spécifié, un de ces outils est utilisé. Rappelons brièvement que l'outil Fujaba ne supporte pas les approches algébriques. Le plugin Viatra est utilisé beaucoup plus dans le domaine de transformation de modèle. AGG se distingue par plusieurs avantages entre autres sa capacité à gérer les approches algébriques et l'API java qu'il fournisse. Dans le cadre de cette thèse, nous avons opté pour l'utilisation de ce dernier pour l'implémentation des règles de réécriture (voir Chapitre 6).

3.5 Applications des grammaires de graphes aux ontologies

Grâce à leurs concepts mathématiques et la formalisation rigoureuse qu'elles fournissent, les grammaires de graphes ont été utilisées dans plusieurs axes de recherche [Ehrig et al., 1996]. Alors que leur premier domaine d'application était le génie logiciel [Schneider, 1997, Mens et al., 2007, da Costa and Ribeiro, 2012], plusieurs autres domaines s'y sont également intéressés. Nous citons à titre d'exemple le domaine de la reconnaissance de formes [Flasiński and Jurek, 2014], de l'imagerie [Lladós and Sánchez, 2003], de la linguistique [Bonfante et al., 2010], etc.

4. user.cs.tu-berlin.de/~gragra/agg

Le domaine des ontologies n'a pas fait l'exception et les premières propositions sont apparues vers l'année 2007 avec les travaux de *d'Aquin et al.* [2007]. Ce couplage entre ontologies et grammaires de graphes continue à intéresser les chercheurs et d'autres travaux sont également proposés, liés essentiellement à la problématique de l'évolution d'ontologies [*Braatz and Brandt*, 2008, *De Leenheer and Mens*, 2008, *Javed et al.*, 2013, *Shaban-Nejad and Haarslev*, 2015]. Cela dit, la plupart de ces travaux se sont focalisés sur la formalisation et les approches proposées n'ont pas été implémentées. Nous classifions ces travaux selon le type de problème qui l'ont traité : (1) travaux qui se sont intéressés à l'interrogation d'ontologies et l'extraction de leurs modules ; (2) travaux qui ont utilisé les concepts de grammaires de graphes pour formaliser des tâches liées à l'évolution d'ontologies ; (3) travaux liés à la problématique de fusion d'ontologies.

3.5.1 Travaux liés à l'interrogation d'ontologies

d'Aquin et al. [2007] ont proposé l'utilisation des transformations de graphes pour extraire des modules (des sous-ensembles d'une ontologie) à partir d'une ontologie. Ils ont défini pour ceci un modèle de représentation d'ontologies composé de quatre nœuds : `Class`, `Property`, `Individual` et `Literal`. Ce modèle est basé sur les graphes attribués et permet d'identifier des modules d'ontologies via des morphismes de graphes. Le travail présente des idées et des perspectives importantes, cependant elles n'étaient pas implémentées.

Braatz and Brandt [2008] se sont intéressés à l'interrogation des ontologies RDF avec le formalisme de grammaire de graphe. Ils ont proposé l'approche MPOC-PO (Minimal PushOut Complement-PushOut) comme extension de l'approche DPO afin de transformer les graphes RDF au formalisme de grammaires de graphes. Le travail introduit la formalisation de différents types de requêtes : sélection, mise à jour (ajout, suppression, etc.) dans le but de proposer un langage d'interrogation plus performant que le langage SPARQL. Cependant, l'approche MPOC-PO n'est pas encore implémentée et par conséquent aucune comparaison détaillée ne peut être élaborée.

3.5.2 Travaux liés à l'évolution d'ontologies

De Leenheer and Mens [2008] ont proposé l'utilisation des grammaires de graphes pour suivre l'évolution des ontologies collaboratives. Le travail s'adresse essentiellement aux ontologies linguistiques. Il définit un modèle de représentation d'ontologies composé de six nœuds : `Term`, `ConceptDefinition`, `ApplicationContext`, `Lexon`, `Genus` et `Deferentia`. Pour résoudre les conflits pouvant résulter de l'évolution parallèle d'ontologies, les auteurs pro-

posent l'utilisation des paires critiques [Plump, 1994]. À noter que les paires critiques sont une technique permettant d'analyser les situations conflictuelles, c'est-à-dire, les cas où l'application d'une transformation empêche l'application d'une autre.

Javed et al. [2013] ont utilisé le formalisme de grammaires de graphe pour la journalisation des changements ontologiques. Les auteurs proposent de représenter les versions d'ontologies et les traces des changements par un graphe attribué. La formalisation de ce graphe a été réalisée par un graphe attribué typé et les changements par une approche TPO (Triple PushOut) [Javed, 2013]. Cette approche a été présentée comme une extension de l'approche DPO et elle consiste à exécuter trois pushout consécutifs. Alors que la formalisation a été effectuée par les approches algébriques, l'implémentation a été réalisée par les techniques de web sémantique en utilisant OWL API⁵.

Shaban-Nejad and Haarslev [2015] ont proposé l'utilisation de la théorie des catégories pour formaliser les changements des ontologies biomédicales. Ils ont défini ainsi quatre catégories : 1) la catégorie *Ontologies* dont les objets sont des ontologies et les morphismes sont des relations entre ontologies ; 2) la catégorie *Class* permet de spécifier les classes d'une ontologie ; 3) la catégorie *State* permet de spécifier l'état d'une ontologie (c'est-à-dire, consistante ou inconsistante) ; 4) la catégorie *Operation* permet de spécifier les opérations sur les entités ontologiques. Les auteurs proposent d'utiliser l'approche DPO pour formaliser les changements ontologiques, sans cependant les implémenter.

3.5.3 Travaux liés à la fusion d'ontologies

À l'heure actuelle et à notre connaissance, aucun travail dans la littérature n'a utilisé les grammaires de graphes dans la problématique de fusion d'ontologies. En effet, les rares travaux proposés dans cet axe de recherche se sont servi uniquement des concepts de la théorie des catégories pour formaliser les opérations de l'alignement et de la fusion. Cela dit, ces travaux sont intéressants et peuvent présenter une source d'inspiration pour les approches algébriques de transformation de graphes qui se basent sur la théorie des catégories. Ceci nous amène à les présenter dans ce manuscrit.

Zimmermann et al. [2006] ont proposé l'utilisation de la théorie des catégories pour formaliser l'opération de l'alignement d'ontologies. Ils ont défini ainsi la catégorie *Onto*⁺ dont les objets sont des ontologies (O_1, O_2, \dots, O_n) et les morphismes sont des fonctions ($f : O_1 \rightarrow O_2$)

5. owlapi.sourceforge.net

permettant d'établir des relations entre des ontologies. Les auteurs ont proposé deux formalismes : le "V-alignment" et le "W-alignment" [Bouquet et al., 2004, Hitzler et al., 2006] basés sur le concept "span" (introduit dans la Section 3.3.1) de la théorie des catégories. Le "V-alignment" consiste à un triplet $\langle O, f_1, f_2 \rangle$ avec : 1) O est une ontologie qui présente la structure commune entre les deux ontologies alignées ; 2) f_1 et f_2 se sont deux morphismes permettant de spécifier la relation entre l'ontologie commune et les deux ontologies alignées. Ce formalisme permet de formaliser les relations d'équivalence par contre il ne peut pas exprimer les relations de subsomption. Pour remédier à ce problème, les auteurs ont proposé le "W-alignment" basé sur deux span consécutifs. Le travail est focalisé principalement sur la formalisation et a ignoré la phase d'implémentation.

Cafezeiro and Haeusler [2007] ont proposé l'utilisation de la théorie des catégories pour formaliser l'opération de la fusion d'ontologies. Le travail se base sur celui de [Zimmermann et al., 2006]. Il définit une catégorie *Ont* dont les objets sont des ontologies (O_1, O_2, \dots, O_n) et les morphismes sont des fonctions permettant d'établir des relations entre des ontologies. Les auteurs utilisent là aussi le concept de "span" pour l'alignement et proposent les concepts de "pushout" et "pullback" pour formaliser l'opération de fusion. Le travail vise essentiellement à formaliser les taxonomies et ne considère que les ontologies composées par des classes et des relations de subsomption. Il ne gère pas les autres relations conceptuelles, ni les axiomes. De plus, aucune implémentation n'est proposée.

3.6 Bilan

Dans ce chapitre, nous avons présenté le formalisme de grammaires de graphes et ses différents concepts. Nous avons étudié les approches de transformation de graphes (ensemblistes et algébriques), leurs caractéristiques et les outils permettant de les implémenter. D'après cette étude, nous avons pu constater la richesse et la capacité de ce formalisme à spécifier différentes situations de transformation de graphes. Ceci présente, effectivement, la raison principale qui a amené plusieurs chercheurs à s'y intéresser. Dans ce chapitre, nous avons présenté particulièrement les travaux qui se sont intéressés au couplage entre ontologies et grammaires de graphes. Les constats que nous pouvons tirer sont que ces travaux sont récents, les plus anciens datent de 2007. Ils sont principalement focalisés sur la formalisation et ne présentent pas, dans la plupart des cas, des implémentations concrètes des problèmes étudiés.

Dans les chapitres qui suivent, nous présenterons nos propositions de couplage d'ontologies et des grammaires de graphes : (1) nous introduirons comment utiliser ce formalisme pour représenter des ontologies OWL (Chapitre 4), (2) comment profiter des concepts de gram-

maires de graphes pour faire évoluer une ontologie d'une manière consistante (Chapitre 4), (3) comment fusionner des ontologies à l'aide de ce formalisme (Chapitre 5) et (4) comment implémenter les changements d'ontologies (évolution et fusion) à l'aide des approches et des outils de transformation de graphes (Chapitre 6).

Deuxième partie :
Propositions

Chapitre 4

Approche d'évolution d'ontologies

Sommaire

4.1 Introduction	83
4.2 Approche d'évolution d'ontologies	85
4.2.1 Modèle de représentation d'ontologies et de leurs changements	86
4.2.2 Processus global d'évolution d'ontologies	88
4.3 Formalisation des changements élémentaires	93
4.3.1 Changements de renommage	95
4.3.2 Changements d'ajout	96
4.3.3 Changements de suppression	97
4.4 Formalisation des changements composés	97
4.4.1 Changements d'ajout	98
4.4.2 Changements de suppression	99
4.5 Formalisation des changements complexes	101
4.5.1 Changements de déplacement	102
4.5.2 Changements de fusion	102
4.5.3 Changements de division	103
4.6 Caractéristiques de l'approche proposée	104
4.7 Bilan	106

*"Tout est changement, tout évolue, tout est en devenir, non pour ne plus être
mais pour devenir ce qui n'est pas encore."
Epictète*

4.1 Introduction

L'évolution d'ontologies est un processus incontournable dans le cycle de vie d'une ontologie. Il consiste à modifier une ou plusieurs de ses entités (classe, propriété, axiome, individus,

etc.) et nécessite une définition d'un ensemble de changements ontologiques capables d'exprimer les différents besoins d'évolution. Faire évoluer une ontologie exige également une formalisation rigoureuse des changements pour garantir la préservation de la consistance de l'ontologie évoluée (voir Chapitre 2, Section 2.4). Ainsi, nous proposons dans notre travail l'utilisation du formalisme de grammaires de graphes typés pour formaliser les changements ontologiques et répondre aux enjeux de la consistance. Nous proposons une nouvelle approche d'évolution d'ontologie permettant de préserver la qualité de l'ontologie évoluée et de résoudre les inconsistances d'une manière a priori. L'approche proposée EvOGG (Evolving Ontologies with Graph Grammars) consiste à représenter les ontologies sous forme de graphes typés attribués. Les changements ontologiques sont formalisés avec les règles de réécriture de graphes : $R(NAC, LHS, RHS)$. Quant à l'évolution d'ontologie, elle est assurée par les transformations de graphes, en particulier par l'approche algébrique Simple PushOut (SPO) (voir Figure 4.1). Trois types de changements ont été considérés : élémentaires (simples), composés et complexes. Les deux niveaux de l'ontologie ont été également traités : (1) l'enrichissement d'ontologies en étudiant les différents changements (ajout, suppression, fusion, etc.) qui affectent les classes, les relations et les axiomes et (2) le peuplement d'ontologies en étudiant les changements qui affectent les individus et leurs assertions.

Ce chapitre vise ainsi à décrire notre approche d'évolution d'ontologie. Nous présentons, en premier lieu, le modèle de transformation de graphes proposé pour la représentation des ontologies et leurs changements. Ensuite, nous détaillons comment notre formalisme représente les différents changements ontologiques et préserve la consistance de l'ontologie évoluée.

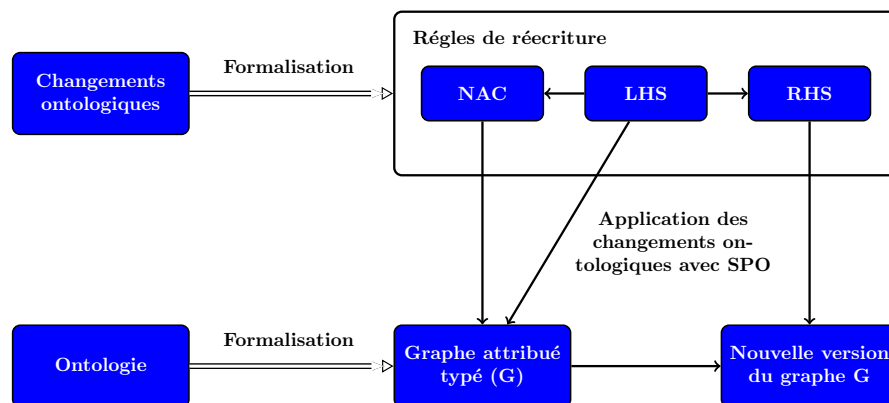


FIGURE 4.1 – Approche proposée pour l'évolution d'ontologies.

4.2 Approche d'évolution d'ontologies

Les chapitres précédents ont montré que les ontologies et les grammaires de graphes possèdent différents points communs. Rappelons premièrement qu'un graphe hôte est une représentation d'un système (ou d'une problématique, d'un processus, etc.) sous forme de nœuds et d'arêtes. Il s'agit d'un modèle qui représente une instance d'un graphe type (le méta-modèle) (voir Chapitre 3, Section 3.2.2). Deuxièmement, les ontologies sont des représentations de connaissances (des modèles) représentées par des langages (des méta-modèles) dont la plupart d'entre eux, notamment le standard OWL, ont des fondements basés sur les graphes (voir Chapitre 2, Section 2.2). Ainsi, représenter une ontologie sous forme d'un graphe typé est cohérent et approprié (voir Figure 4.2).

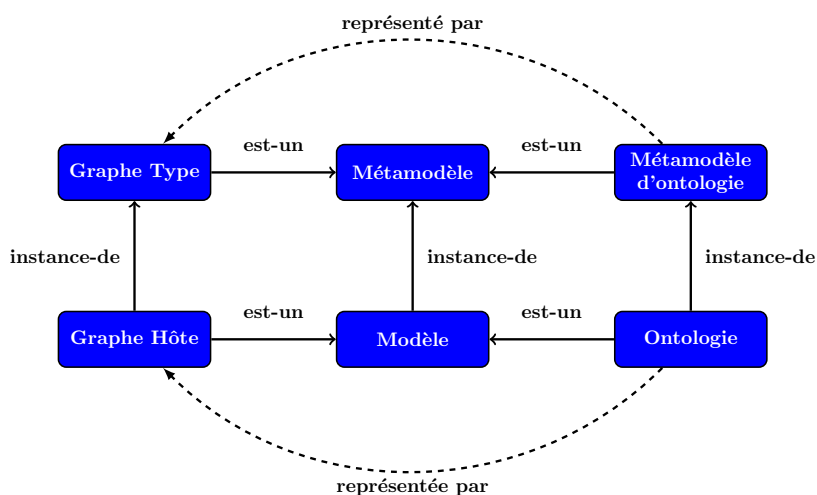


FIGURE 4.2 – Relation entre ontologies et grammaires de graphes typés.

Nous décrivons dans ce qui suit notre modèle de représentation d'ontologies et des changements ontologiques avec les grammaires de graphes typés. Nous présentons aussi le processus général de notre approche d'évolution d'ontologies EvOGG (Evolving Ontologies with Graph Grammars) et ses différentes étapes : (1) l'identification de changement ; (2) la formalisation de changement et (3) l'implémentation de changement.

4.2.1 Modèle de représentation d'ontologies et de leurs changements

Afin de représenter les ontologies et les changements ontologiques avec les grammaires de graphes typés, nous proposons le modèle TGGOnto (Typed Graph Grammars for Ontologies) suivant :

$$TGGOnto = \{GT_O, G_O, R_O\}, \text{ avec}$$

GT_O représentant le graphe type (le méta-modèle d'une ontologie), G_O est le graphe hôte représentant une ontologie (une instance du graphe type) et R_O sont les règles de réécritures décrivant les changements ontologiques.

Pour représenter le graphe type, nous avons retenu le méta-modèle OWL dans la mesure où c'est le langage communément adopté pour représenter les ontologies. Nous nous concentrons, ainsi, sur l'évolution des ontologies OWL et leurs axiomes issus des logiques de description SHOIN(D) (voir Chapitre 2, Section 2.2.1). La Figure 4.3 présente le graphe type que nous proposons qui est un graphe attribué et étiqueté.

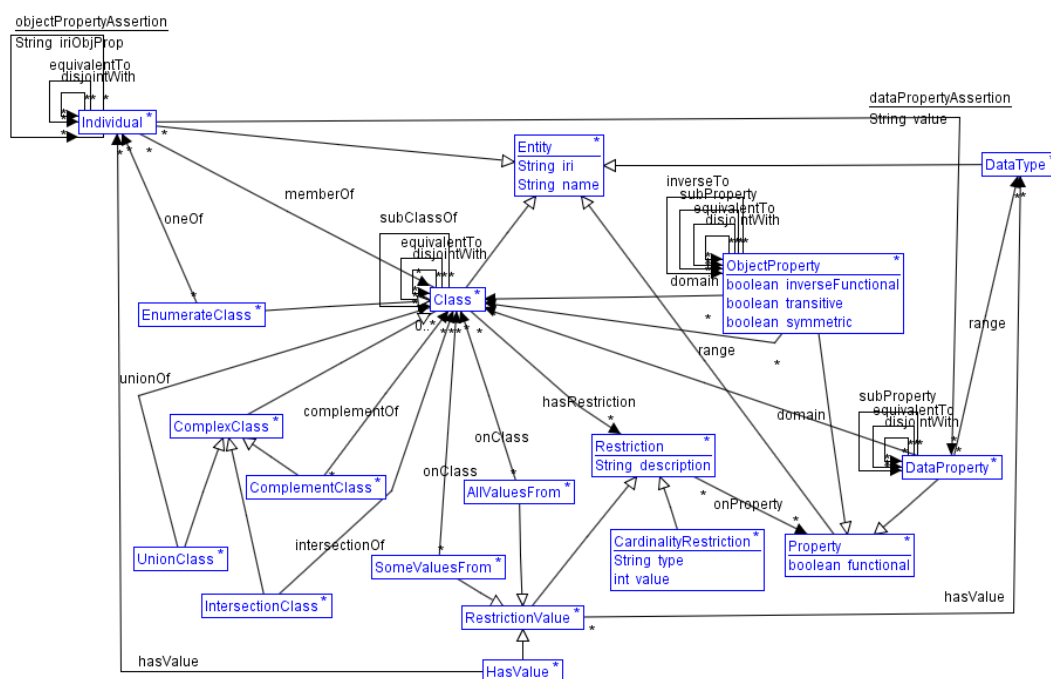


FIGURE 4.3 – Graphe type utilisé pour décrire les ontologies.

Les types des nœuds considérés sont :

$$N_T = \{Class(C), Property(P), ObjectProperty(OP), DataProperty(DP), Individual(I), DataType(D), Restriction(R), \dots\}.$$

Les types des arêtes correspondent aux axiomes utilisés pour relier les différentes entités :

$$E_T = \{subClassOf, equivalentTo, range, domain, \dots\}.$$

En effet, les classes modélisent l'ensemble des individus (Individual) et peuvent être primitives (Class) ou bien complexes (UnionClass, ComplementClass, IntersectionClass). Les propriétés (Property) peuvent être de deux types : 1) les ObjectProperty modélisent des relations entre des classes et 2) les DataProperty relient une classe à une valeur typée (Datatype). Tous ces nœuds possèdent deux attributs hérités du nœud Entity. L'attribut name précise le nom local de l'entité alors que l'attribut iri permet de les identifier et de les référencer d'une manière unique. Chacun de ces nœuds possède ses propres attributs. Par exemple, nous identifions pour les propriétés, l'attribut functional qui précise si une propriété est fonctionnelle. Deux types de restrictions sont représentés : les restrictions de valeur (AllValuesFrom, SomeValuesFrom, HasValue) et les restrictions de cardinalité (CardinalityRestriction). Les axiomes sont représentés sous forme d'arêtes exprimant les relations entre classes, propriétés et individus. Par exemple, l'arête disjointWith est utilisée pour exprimer la disjonction entre deux classes ou encore entre deux propriétés. L'arête subClassOf représente la relation de subsomption entre deux classes. Les arêtes à leur tour peuvent être attribuées. Nous distinguons, par exemple, l'arête dataPropertyAssertion qui possède l'attribut value. La Figure 4.4 présente un exemple d'ontologie (un graphe hôte) représentée selon le formalisme proposé.

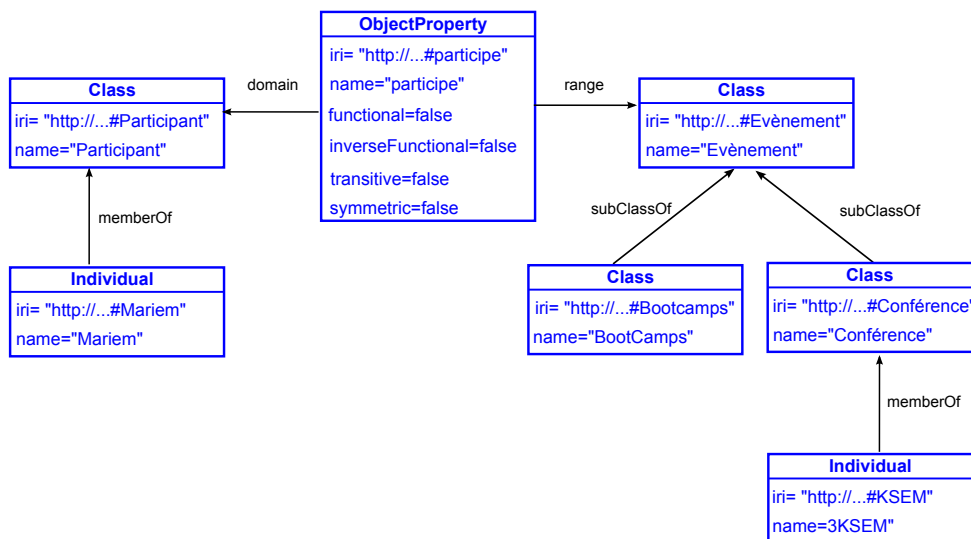


FIGURE 4.4 – Exemple d'ontologie conforme au graphe type de la Figure 4.3.

4.2.2 Processus global d'évolution d'ontologies

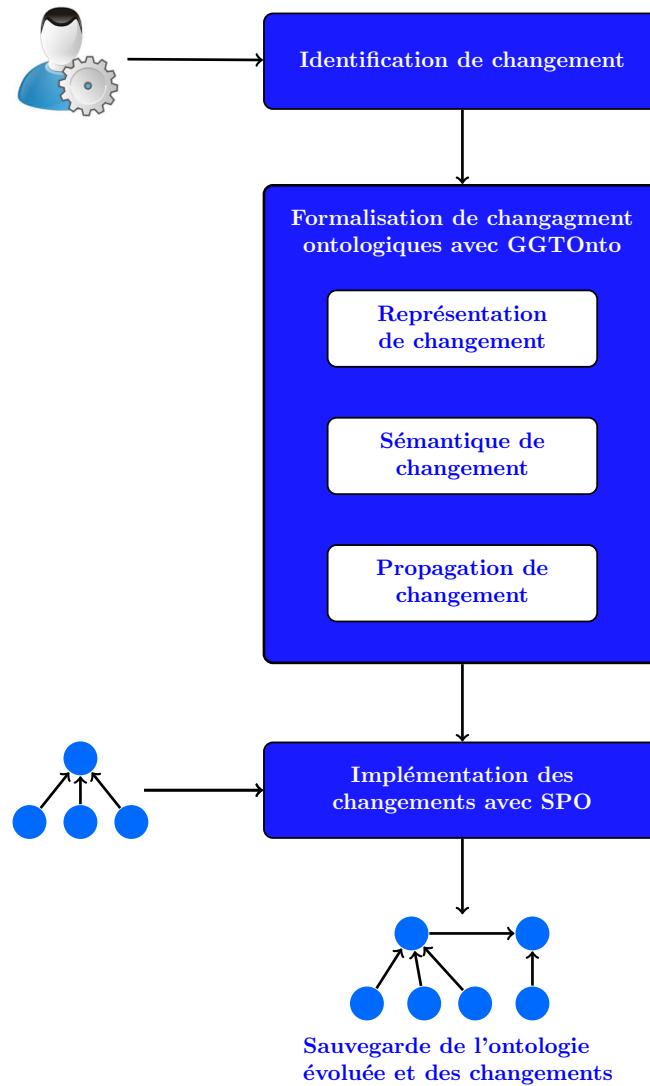


FIGURE 4.5 – Processus global de l'approche d'évolution d'ontologies EvOGG.

Nous proposons un processus d'évolution d'ontologies composé de trois grandes étapes (Figure 4.5) : (1) l'*identification de changement* par l'utilisateur, (2) la *formalisation de changement* avec les règles de réécriture de graphes et (3) l'*implémentation des changements* avec l'approche algébrique Simple Pushout (SPO).

Identification de changement

L'étape d'*identification de changement* permet de déterminer les changements ontologiques nécessaires pour exprimer les besoins d'évolution (voir Chapitre 2, Section 2.4.3). D'une manière générale, cette étape peut être réalisée soit manuellement par l'utilisateur ou bien semi-automatique en faisant appel aux techniques de fouille de données et de traitement automatiques des langues (TAL). Dans le deuxième cas, les besoins d'évolution sont capturés à partir de l'analyse des versions d'ontologies et/ou de l'analyse des profils d'utilisateurs. Dans notre travail, c'est l'utilisateur qui se charge de choisir les changements qu'il voulait apporter à l'ontologie.

Formalisation de changement avec les règles de réécriture

L'étape de *formalisation de changement* permet de représenter les changements dans un langage approprié tout en préservant la consistance de l'ontologie évoluée. Elle englobe :

1. La *représentation de changement* qui consiste à représenter les changements identifiés dans la phase précédente avec le formalisme de grammaire de graphe et les règles de réécriture de graphes (voir Sections 4.3, 4.4 et 4.5).
2. La *sémantique de changement* qui permet de maintenir la consistance et la cohérence de l'ontologie. Sachant que dans notre travail, nous nous plaçons dans le cadre des approches de résolution a priori des inconsistances, la sémantique de changement est prise en compte lors de la représentation de changement. Elle est assurée lors de la formalisation des règles de réécriture, essentiellement, par les conditions d'applications négatives (NAC). Plus de détails dans les Sections 4.3, 4.4 et 4.5.
3. La *propagation de changement* qui permet de propager les changements vers les artéfacts dépendants. En général, les artéfacts peuvent être de trois catégories : 1) les instances de l'ontologie ; 2) des ontologies reliées à l'ontologie évoluée ; 3) des sources externes utilisant l'ontologie (par exemple des dictionnaires, des thésaurus, etc.). Actuellement, nous nous intéressons à la première catégorie et la propagation est assurée lors de la formalisation des règles de réécriture (voir Sections 4.4 et 4.5).

Nous distinguons dans notre travail trois types de changements : élémentaires, composés et complexes. Rappelons que cette classification a été proposée par *Stojanovic* [2004] pour distinguer les changements d'ontologies KAON. Nous l'adaptions pour les ontologies OWL tout en ajoutant les changements correspondant à ce langage (par exemple les changements de cardinalité, les assertions sur les individus, etc.). La classification des changements abordés sont présentés par la Figure 4.6.

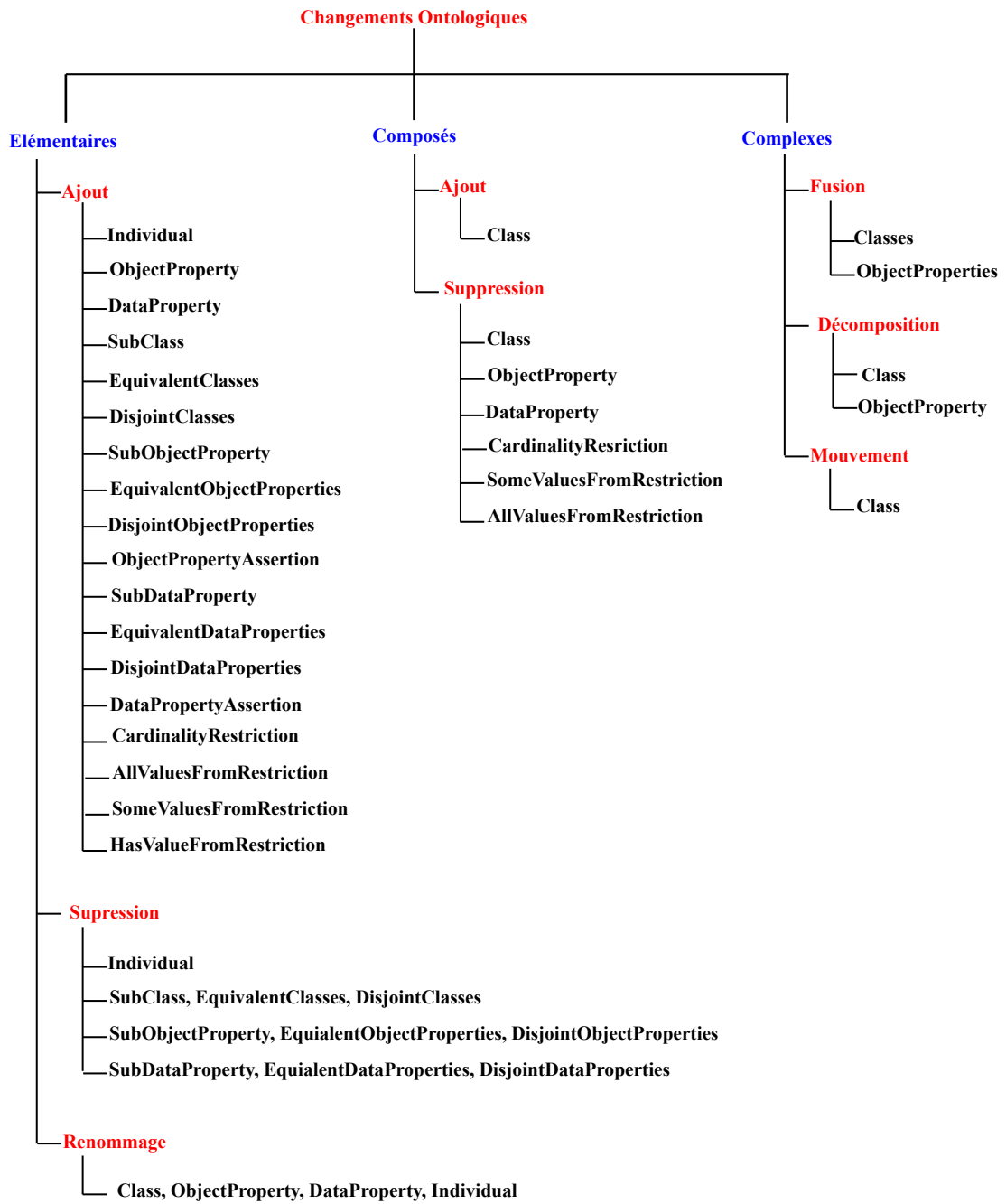


FIGURE 4.6 – Taxonomie des changements étudiés.

Comme déjà évoqué, les changements ontologiques (*CH*) sont formalisés dans notre modèle $TGGOnto = \{GT_O, G_O, R_O\}$ avec les règles de réécriture de graphes $R_O = \{r_1, r_2, \dots, r_i, \dots\}$. Une règle r_i est définie comme suit :

$$r_i = (NACs, LHS, RHS, CHDs), \text{ avec :}$$

- le *LHS* est un graph pattern qui représente les pré-conditions d'un changement ontologique (c'est-à-dire, ce qui doit être vrai pour que le changement puisse être appliqué) ;
- le *RHS* est un graph pattern qui définit le changement apporté à l'ontologie ;
- les *NACs* sont des graphes patterns définissant les conditions qui ne doivent pas être vraies pour pouvoir appliquer le changement ontologique. Elles permettent d'empêcher l'application de tout changement qui ne satisfait pas les conditions qu'elles définissent. Nous nous servons de ces conditions pour assurer une résolution a priori des inconsistances ;
- les *CHDs* = (*NACs*, *LHS*, *RHS*) présentent les changements dérivés. Ce sont des règles de réécriture additionnelles attachées au CH (le changement principal) pour corriger ses éventuelles inconsistances et préserver la consistance de l'ontologie évoluée. Elles permettent également de propager les changements aux artéfacts dépendants.

Nous nous intéressons dans notre travail à la consistance interne d'une ontologie, essentiellement la cohérence logique et conceptuelle (voir Chapitre 2, Section 2.3.3). Ainsi, les inconsistances et les incohérences sont essentiellement :

- La *redondance de données* qui peut être générée suite à un changement d'ajout ou de renommage. Nous présenterons dans les sections ci-après comment elle peut être corrigée par les *NACs*.
- Les *nœuds isolés* qui peuvent être le résultat d'un changement d'ajout ou de suppression. Un nœud N_x est dit isolé ssi $\forall N_i \in N, \nexists E_i \in E | E_i = (N_x \times N_i)$. Cette incohérence nécessite de rattacher le N_x au reste du graphe et selon son type, des changements dérivés sont proposés.
- Les *individus orphelins* qui résultent d'un changement de suppression des classes définissantes des individus. Pour empêcher cette incohérence, un ensemble de changements dérivés est proposé.
- Les *axiomes contradictoires* qui peuvent être générés suite à un changement d'ajout. En effet, l'ajout d'un nouvel axiome ne doit pas être accepté s'il va contredire les axiomes déjà définis dans l'ontologie. De nombreux cas sont considérés : deux classes ne peuvent pas être à la fois disjointes et équivalentes, deux classes qui partagent une relation de subsomption ne peuvent pas être disjointes, etc. Ce type d'inconsistance est géré par les *NACs*.

Implémentation des changements avec l'approche SPO

L'étape d'*implémentation des changements* consiste à appliquer les changements ontologiques, sauvegarder la nouvelle version de l'ontologie évoluée et garder une trace des changements appliqués. Étant représentés avec les règles de réécriture de graphe, l'implémentation des changements ontologiques sera réalisée par une des approches de transformation de graphes. Dans notre travail, nous avons choisi d'utiliser l'approche algébrique Simple Pushout (SPO). Nous justifions dans ce qui suit ce choix.

Comme déjà introduit dans le Chapitre 3 (Section 3.2.2), deux types de familles d'approches sont généralement utilisées pour les transformations de graphes : 1) les approches ensemblistes (essentiellement les approches "node replacement" et "edge replacement") et 2) les approches algébriques. Les approches ensemblistes souffrent de plusieurs limites, notamment leur incapacité à gérer les graphes attribués et leur ignorance du contexte (les transformations ne peuvent s'appliquer que sur une seule entité atomique, soit un nœud ou bien une arête). C'est à cause de ces limites que ces approches ne peuvent pas être appliquées dans le cadre des ontologies. En effet, d'une part, les ontologies sont représentées sous forme de graphes attribués. D'autre part, les changements ontologiques, notamment les changements complexes et composés prennent en compte le contexte des entités.

En revanche, les approches algébriques sont plus expressives. Elles utilisent des graphes attribués et typés et permettent de transformer des graphes entiers. Elles offrent deux types d'approches, l'approche Simple PushOut (SPO) et l'approche Double PushOut (DPO). Dans notre travail, c'est l'approche SPO qui a été retenue car elle se voit plus générale et permet d'appliquer les différents changements ontologiques étudiés (voir Figure 4.6). L'approche DPO, par contre, est limitée par la condition de "gluing condition" et elle n'autorise pas l'application de certains changements, notamment ceux de suppression.

La Figure 4.7 montre la représentation et l'application de la règle de réécriture du changement ontologique *AddIndividual* par l'approche SPO. La règle permet d'ajouter un nouveau individu "Nibel" à la classe "Participant" d'une ontologie (G) et elle est définie comme suit :

- $NAC = \{I_{Nibel}\}$. Pour éviter la redondance de données, le *NAC* de cette règle de réécriture doit être un graphe composé d'un nœud de type `Individual` avec l'attribut `name="Nibel"`. Ceci empêche l'application du changement dans le cas où l'individu existe déjà dans l'ontologie.
- $LHS = \{C_{Participant}\}$. Le *LHS* représente la pré-condition de la règle. Il est alors, dans ce cas, un graphe composé par un nœud de type `Class` avec `name="Participant"`. Il permet de vérifier si la classe dont on veut ajouter l'individu existe déjà dans l'ontologie.
- $RHS = \{C_{Participant}, I_{Nibel}\}$. Le *RHS* spécifie le nouveau graphe qui va remplacer le LHS dans l'ontologie. Il est composé par les deux nœuds $C_{Participant}$ et I_{Nibel} reliés par l'arête

memberOf pour spécifier la relation entre eux.

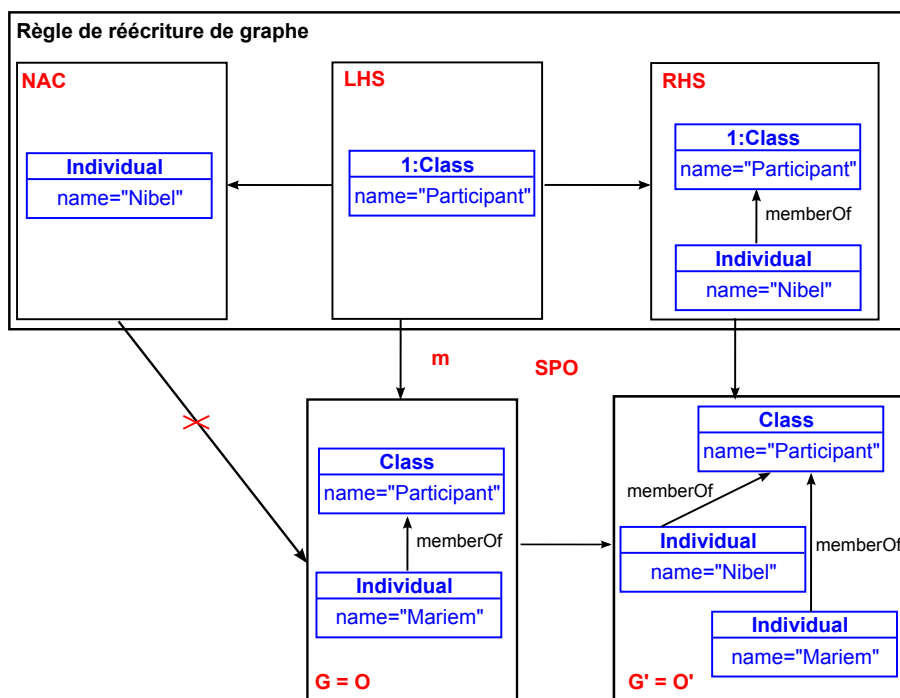


FIGURE 4.7 – Représentation et application de la règle de réécriture du changement *AddIndividual* avec l'approche SPO.

Dans ce qui suit, nous présentons la formalisation des changements ontologiques (élémentaires, composés et complexes) et nous montrons comment le formalisme de grammaires de graphes typés est capable d'éviter les inconsistances et de préserver par conséquent la qualité de l'ontologie évoluée.

4.3 Formalisation des changements élémentaires

Les changements élémentaires englobent les changements de renommage, d'ajout et de suppression de certains concepts. Ils n'affectent qu'une seule entité ontologique bien qu'ils dépendent d'autres entités. Le Tableau 4.1 présente les changements élémentaires considérés dans notre travail et les concepts dont ils dépendent. À noter que les *NACs* des règles de réécriture sont déduites à partir de ces interdépendances. Par exemple, à partir de ce tableau, nous pouvons constater que le changement *AddDataPropertyAssertion(I, DP, value)*, qui ajoute

4.3. Formalisation des changements élémentaires

une `DataPropertyAssertion` entre un individu `I` et une `dataProperty` `DP`, dépend des entités `Individual`, `DataProperty` et `FunctionalProperty`. En effet, avant d'appliquer ce changement, il est nécessaire de vérifier si la `dataProperty` `DP` est une propriété fonctionnelle. Dans ce cas, si l'individu `I` possède déjà une `AssertionDataProperty` avec `DP`, alors, il faut interdire l'application du changement pour qu'il n'affecte pas la consistance de l'ontologie.

	Class	Individual	ObjectProperty	DataProperty	Data Type	EquivalentClass	DisjointClass	SubClass	EquivalentProperty	DisjointProperty	SubProperty	FunctionalProperty	CardinalityRestriction	AllValuesFromRestriction	SomeValuesFromRestriction	HasValueRestriction
RenameClass	✓															
RenameIndividual		✓														
RenameObjectProperty			✓													
RenameDataProperty				✓												
AddIndividual	✓	✓														
AddDataProperty	✓			✓	✓											
AddObjectProperty	✓		✓													
AddEquivalentClasses	✓	✓				✓	✓									
AddDisjointClasses	✓	✓				✓	✓	✓								
AddSubClass	✓	✓					✓	✓								
AddObjectPropertyAssertion		✓	✓									✓	✓			
AddDataPropertyAssertion		✓		✓								✓				
AddSubObjectProperty			✓							✓	✓					
AddSubDataProperty				✓						✓	✓					
AddEquivalentObjectProperties			✓						✓	✓						
AddDisjointObjectProperties			✓						✓	✓						
AddEquivalentDataProperties				✓					✓	✓						
AddDisjointDataProperties				✓					✓	✓						
AddCardinalityRestriction	✓	✓	✓										✓			
AddAllValuesFromRestriction	✓	✓											✓			
AddSomeValuesFromRestriction	✓	✓	✓												✓	
AddHasValueRestriction	✓		✓													✓
RemoveIndividual		✓														
RemoveDisjointClasses	✓						✓									
RemoveEquivalentClasses	✓					✓										
RemoveSubClass	✓							✓								
RemoveEquivalentObjectProperties			✓						✓							
RemoveDisjointObjectProperties			✓							✓						
RemoveSubObjectProperties			✓								✓					
RemoveSubDataProperties				✓							✓					
RemoveEquivalentDataProperties				✓					✓							

RemoveDisjointDataProperties					✓					✓					
------------------------------	--	--	--	--	---	--	--	--	--	---	--	--	--	--	--

TABLE 4.1: Matrice de dépendance entre les changements élémentaires et les entités ontologiques.

Dans ce qui suit, nous présentons la formalisation d'une sélection de changements de renommage, d'ajout et de suppression. Les autres changements sont décrits dans l'Annexe A. A noter que les changements élémentaires ne possèdent pas des changements additionnels (*CHD*). Pour certains changements aucun *NAC* n'est défini puisque leur application ne peut causer aucune inconsistance sur l'ontologie.

4.3.1 Changements de renommage

Les changements de renommage concernent les classes, les individus et les propriétés. Nous présentons dans cette section la règle de réécriture correspondante au changement *RenameIndividual*(I_i, I_{New}) qui renomme un nœud de type *Individual*. La règle de réécriture est défini comme suit (Figure 4.8) :

- *NAC* = $\{I_{New}\}$. Pour éviter la non redondance de données, le *NAC* de cette règle doit être un graphe composé d'un nœud de type *Individual* avec l'attribut *name* = I_{New} . Ceci signifie que ce sous-graphe ne doit pas exister dans le graphe de l'ontologie pour que le changement puisse être appliqué.
- *LHS* = $\{I_i\}$. Le *LHS* est un graphe composé par un nœud de type *Individual* avec l'attribut *name* = I_i . Ainsi, pour pouvoir appliquer le changement, le nœud doit déjà exister dans l'ontologie.
- *RHS* = $\{I_{New}\}$. Le graphe *RHS* spécifie le nouveau graphe qui doit remplacer *LHS* et précise ainsi la nouvelle valeur de l'attribut.

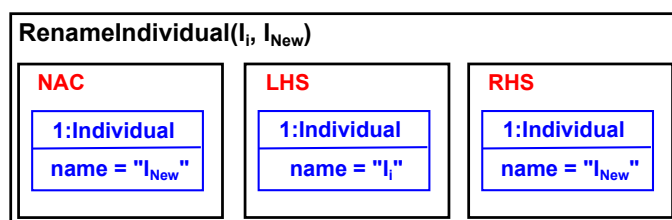


FIGURE 4.8 – Règle de réécriture du changement *RenameIndividual*.

4.3.2 Changements d'ajout

Les changements d'ajout concernent certains concepts (les individus et les propriétés), les axiomes (`subClassOf`, `disjointOf`, `ObjectPropertyAssertion`, etc.) et les restrictions (`CardinalityRestriction`, `AllValuesRestriction`, etc.). À titre d'exemple, nous présentons dans cette section la règle de réécriture correspondante au changement *AddDisjointClasses* (C_1 , C_2) qui permet d'ajouter un lien de disjonction entre deux nœuds de type `Class`. La Figure 4.9 présente la règle qui est formalisée comme suit :

– *NACs* :

1. $C_1 \sqsubseteq \neg C_2$, condition pour éviter la redondance ;
2. $C_1 \equiv C_2$, deux classes ne peuvent pas être à la fois équivalentes et disjointes ;
3. $C_1 \sqsubseteq C_2$, deux classes partageant une relation de subsomption ne peuvent pas être disjointes ;
4. $C_2 \sqsubseteq C_1$, deux classes partageant une relation de subsomption ne peuvent pas être disjointes ;
5. $\exists I_i \in I(O) \cdot I_i \in C_1 \wedge I_i \in C_2$, deux classes disjointes ne peuvent pas avoir des individus communs.

– *LHS* = $\{C_1, C_2\}$, les deux classes doivent exister dans l'ontologie.

– *RHS* = $\{C_1 \sqsubseteq C_2\}$, l'axiome est ajouté à l'ontologie.

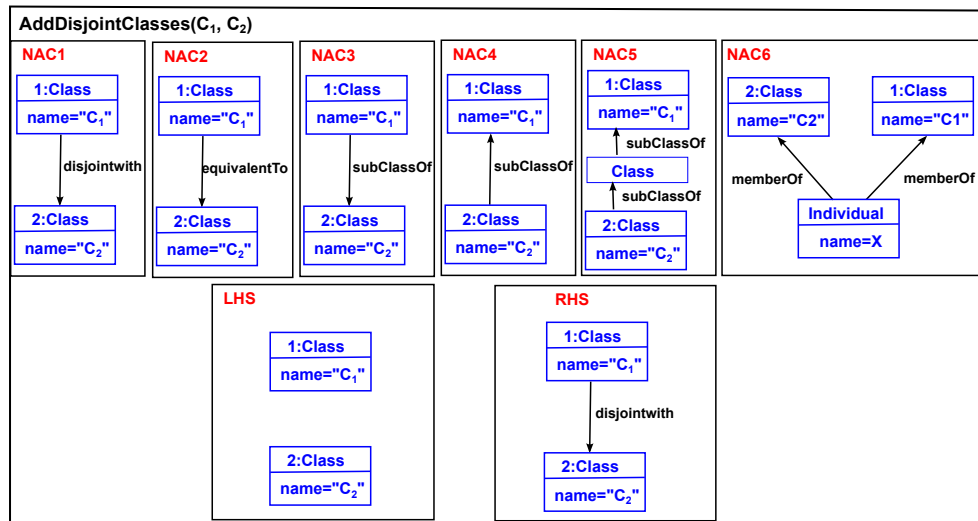


FIGURE 4.9 – Règle de réécriture du changement *AddDisjointClasses*.

4.3.3 Changements de suppression

Les changements élémentaires de suppression englobent la suppression des individus et des axiomes (*subClass*, *equivalentClasses*, *disjointObjectProperties*, etc.). Nous présentons dans ce qui suit l'exemple de changement *RemoveEquivalentObjectProperties*(OP_1, OP_2). Ainsi, la règle de réécriture correspondant à ce changement supprime l'axiome *equivalentTo* entre deux objectProperty (Figure 4.10) et elle est définie par :

- $NAC = \emptyset$
- $LHS = \{OP_1 \equiv OP_2\}$, les objectProperties et leur relation d'équivalence doivent exister dans l'ontologie.
- $RHS = \{OP_1, OP_2\}$, l'axiome doit être supprimé de l'ontologie.

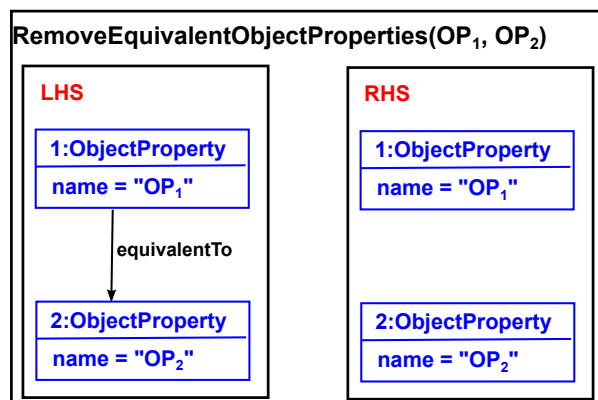


FIGURE 4.10 – Règle de réécriture du changement *RemoveEquivalentObjectProperties*.

4.4 Formalisation des changements composés

Les changements ontologiques composés, appelés aussi composites, affectent une entité ontologique et ses voisins. Ils sont formés par plusieurs règles de réécriture : une règle présentant le changement souhaité par l'utilisateur (changement principal) et les règles présentant les changements dérivés (*CHD*) pour préserver la consistance de l'ontologie. À noter que l'ordre des règles de réécriture (*CH* et *CHDs*) est primordial dans la plupart des changements pour gérer les cas des *paires critiques*. Rappelons qu'une paire critique est une paire de transformations parallèlement dépendantes, soient $r_1 : G_1 \rightarrow G$ et $r_2 : G_2 \rightarrow G$ deux règles de réécriture possédant le même codomaine (un graphe G).

Le Tableau 4.2 présente l'interdépendance entre les changements composés organisés dans une matrice de changements. La valeur d'un élément de matrice (i, j) indique que l'application d'un changement relié à une ligne i implique l'application du changement de la colonne j . Nous présentons dans ce qui suit la formalisation de certains changements.

	AddClass	RemoveClass	AddSubClass	AddDisjointClasses	AddEquivalentClasses	RemoveIndividual	AddTypeIndividual	RemoveTypeIndividual	RemoveAssertionObjectProperty	RemoveCardinalityRestriction	RemoveAllValuesFromRestriction	RemoveSomeValuesFromRestriction	RemoveHasValueRestriction
AddClass	✓		✓	✓	✓		✓						
RemoveClass		✓				✓	✓	✓	✓	✓	✓	✓	✓
RemoveObjectProperty									✓	✓	✓	✓	
RemoveDataProperty									✓	✓	✓	✓	
RemoveCardinalityRestriction									✓	✓			
RemoveSomeValuesFromRestriction									✓			✓	
RemoveAllValuesFromRestriction									✓		✓		

TABLE 4.2: Matrice de dépendance entre les changements ontologiques composés.

4.4.1 Changements d'ajout

Dans notre travail, seul le changement d'ajout d'une classe ($AddClass(C_{New})$) est considéré comme changement composé. Les autres changements d'ajout sont plutôt classifiés comme des changements simples (plus de détails dans la Section 4.6). Le changement d'ajout d'une classe peut être considéré d'une première vue, comme un changement simple. Cependant, et afin de préserver la consistance conceptuelle d'une ontologie, il faut s'assurer lors de l'application d'un changement de ne pas avoir un concept isolé. Ainsi, l'ajout d'une classe doit être accompagné par des changements additionnels permettant son rattachement au reste de l'ontologie. Deux types d'alternatives de correction pouvant être appliquées : $AddObjectProperty$ et $AddAxiom$. La première consiste à ajouter à l'ontologie une nouvelle propriété dont le nœud C_{new} est l'un de ses deux membres. La deuxième alternative consiste à ajouter un nouvel axiome permettant de relier C_{new} à une propriété déjà existante ($AddDomain$, $AddRange$) ou bien le relier à un autre nœud de type $Class$ et ceci en appliquant les changements $AddEquivalentClasses$, $AddSubClass$, etc. Ces alternatives seront proposées à l'utilisateur

et c'est à lui de décider quelle alternative veut appliquer. La Figure 4.11 présente la formalisation de changement d'ajout d'une classe avec l'alternative de correction *AddEquivalentClasses*.

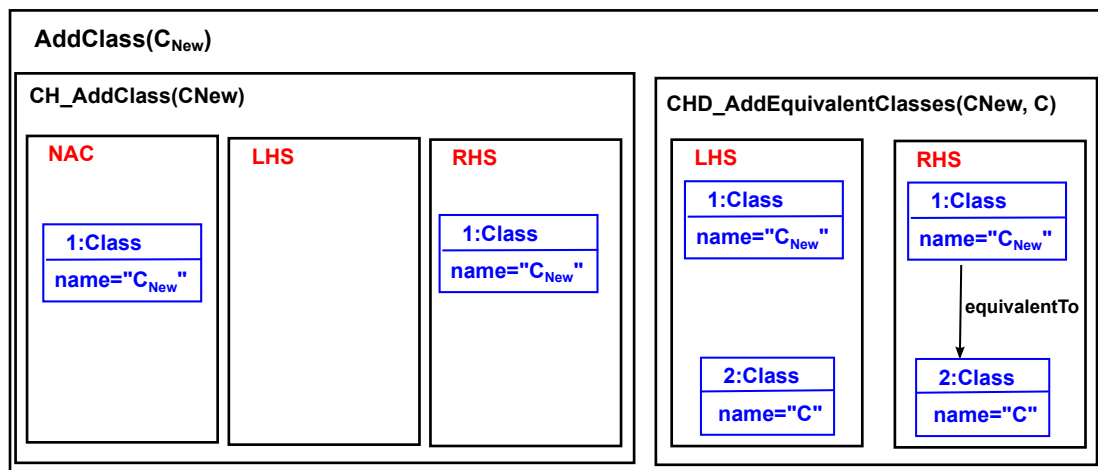


FIGURE 4.11 – Règle de réécriture du changement *AddClass* avec l'alternative de correction *AddEquivalentClasses*.

4.4.2 Changements de suppression

Les changements composés de suppression englobent essentiellement la suppression des classes, des propriétés et des restrictions. Nous présentons dans ce qui suit un exemple de chacun.

RemoveCardinalityRestriction. Le changement *RemoveCardinalityRestriction(C, OP)* permet de supprimer une *CardinalityRestriction* définie sur une classe *C* et une object-Property *OP*. Il est formalisé par deux règles de réécriture. La première représente le changement dérivé *RemoveAssertionObjectProperty* qui supprime toutes les assertions définies sur *OP*. La deuxième règle définit la règle de réécriture principale assurant la suppression de la restriction.

RemoveObjectProperty. Le changement ontologique *RemoveObjectProperty(OP)* supprime une objectProperty *OP* et toutes ses dépendances de l'ontologie. La Figure 4.12 présente

les six règles de réécriture définissant ce changement. Ainsi, les cinq premières règles décrivent les changements dérivés (*CHD*) devant être appliqués pour préserver la consistance de l'ontologie. La dernière règle présente la règle de réécriture principale. Ainsi, les restrictions définies sur la propriété *OP* doivent être supprimées en appliquant les règles suivantes : *RemoveAllValuesRestriction(OP)*, *RemoveSomeValuesRestriction(OP)*, *RemoveHasValueRestriction(OP)* et *RemoveCardinalityRestriction(OP)*. Toutes les *ObjectPropertyAssertion* qui référencent l'*ObjectProperty OP* doivent également être supprimées.

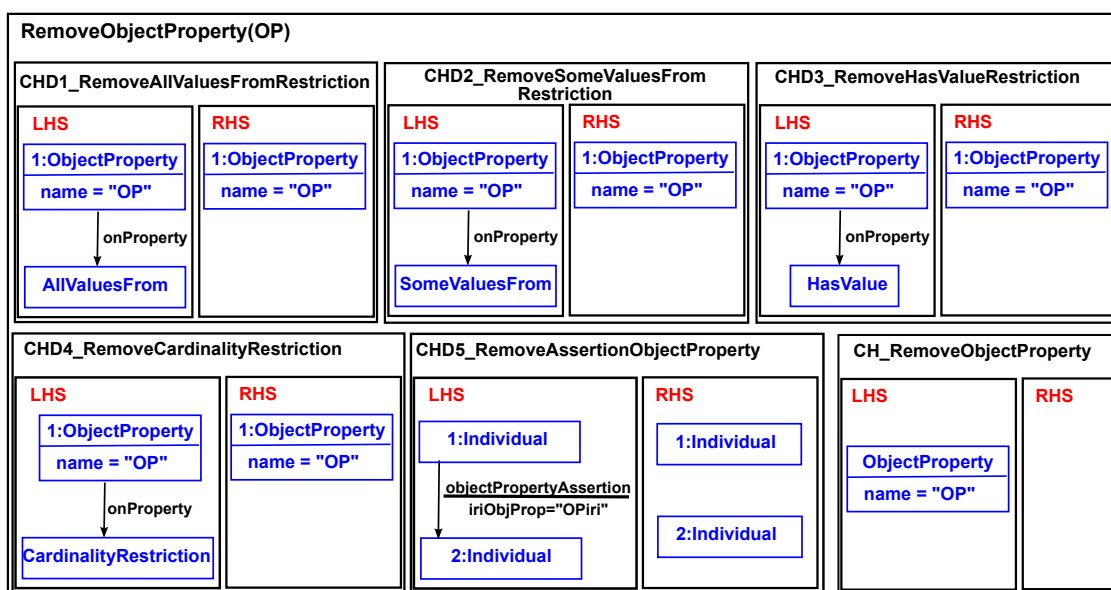


FIGURE 4.12 – Règle de réécriture du changement *RemoveObjectProperty*.

RemoveClass. Le changement *RemoveClass(C)* permet de supprimer un nœud de type *Class* de l'ontologie. Son application peut engendrer certaines incohérences comme par exemple, l'existence des individus orphelins ou bien encore le manque des membres d'une contrainte de restriction. A partir de là, avant de supprimer un nœud, il faut vérifier toutes ses dépendances pour en proposer des corrections. En effet, les restrictions devront être supprimées alors que le traitement des individus passe par différentes étapes (Figure 4.13). Ainsi, avant de supprimer une classe *C* possédant des individus *I*, il faut vérifier que : 1) si $\exists C_i \in G$ tel que C_i *equivalentTo* *C*, alors, I *memberOf* C_i ; 2) sinon si $\exists I_i \in G$ tel que I_i *memberOf* $C_i \wedge I_i$ *equivalentTo* *I*, alors, I *memberOf* C_i ; 3) sinon si C *subclassOf* $C_p \wedge \forall C_i$ *subclassOf* C_p

$\wedge !disjointWith C$, alors, $I \text{ memberOf } C_p$; 4) si aucun de ces cas n'est satisfait les individus orphelins seront supprimés.

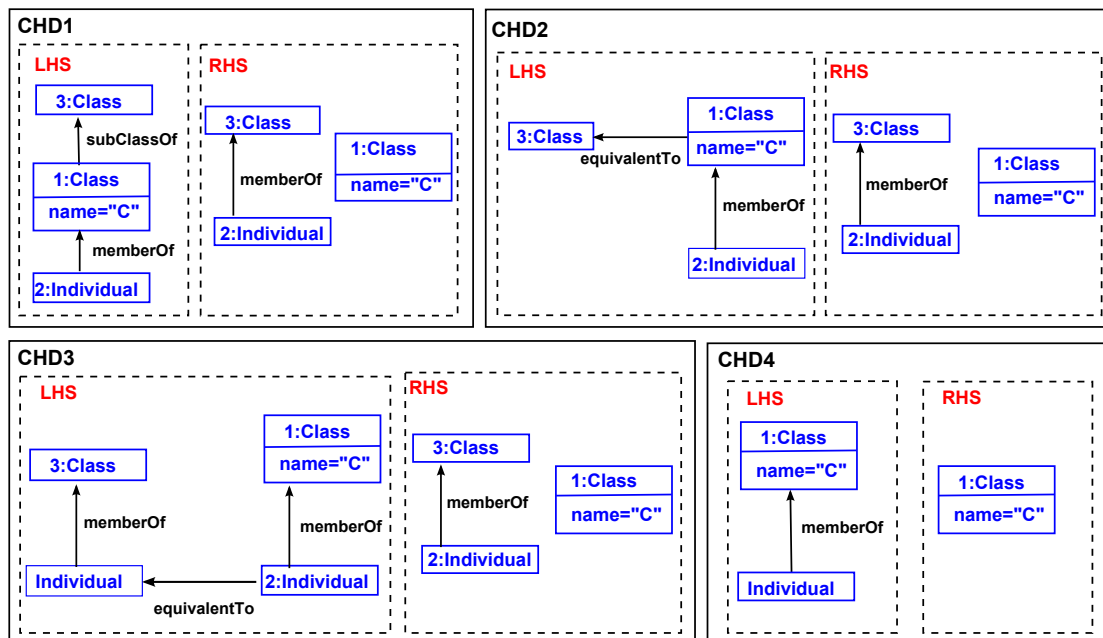


FIGURE 4.13 – Changements dérivés `RemoveIndividual` du changement `RemoveClass`.

4.5 Formalisation des changements complexes

Les changements ontologiques complexes regroupent plusieurs changements élémentaires et composés et affectent des entités ontologiques qui ne sont pas nécessairement adjacentes. Ils sont principalement définis pour agréger différents changements dans une seule opération. Ils sont très utiles dans le sens où ils aident l'utilisateur à adapter son ontologie sans se perdre dans les détails des changements élémentaires. Cependant, ils cachent derrière une formalisation sophistiquée puisqu'ils affectent, à la fois, plusieurs entités ontologiques et peuvent causer des inconsistances à l'ontologie évoluée.

Le Tableau 4.3 présente l'ensemble des changements complexes abordés dans ce travail et les changements dont ils sont composés.

	AddClass	RemoveClass	AddSubClass	AddDisjointClasses	AddEquivalentClasses	AddIndividual	AddObjectProperty	RemoveObjectProperty	RemoveObjectPropertyAssertion	RemoveDataPropertyAssertion	AddSubProperty	AddEquivalentProperties	AddDisjointProperties
PullUpClass			✓						✓	✓			
MergeClasses	✓	✓	✓	✓	✓	✓							
SplitClass	✓	✓	✓	✓	✓	✓							
SplitObjectProperty							✓	✓			✓	✓	✓
MergeObjectProperties							✓	✓			✓	✓	✓

TABLE 4.3: Matrice de dépendance des changements ontologiques complexes.

4.5.1 Changements de déplacement

D'une manière générale, les changements de déplacement concernent les classes (*PullUpClasses* et *PullDownClasses*). Ces deux changements sont considérés dans la littérature comme changements complexes (ou bien composés selon la classification adoptée). Dans notre travail, seul le changement *PullUpClasses* est considéré comme complexe. Le deuxième est plutôt un changement simple formalisé par une seule règle de réécriture (voir Section 4.6).

Ainsi, le changement $PullUpClass(C, C_p)$ permet de monter une classe C dans sa hiérarchie de classes et l'attacher aux parents de sa super-classe précédente C_p . Ceci implique que la classe C n'est plus la *subClass* de la classe C_p et n'infère plus ses propriétés. La Figure 4.14 présente la règle de réécriture définissant ce changement. Ainsi, le changement dérivé *RemoveObjectPropertyAssertion* vérifie si la classe C possède des individus qui partagent une *objectPropertyAssertion* sur les propriétés de la classe C_p . Dans ce cas, toutes les assertions doivent être supprimées. Le changement *RemoveDataPropertyAssertion* supprime toutes les *dataPropertyAssertion* définies sur les individus de la classe C et les *dataProperties* liées à la classe C_p .

4.5.2 Changements de fusion

Les changements de fusion concernent les classes et les propriétés. Nous présentons dans cette section l'exemple de fusion de deux classes. Ainsi, le changement $MergeClasses(C_1, C_2, C_{New})$ fusionne deux classes C_1 et C_2 déjà existantes dans l'ontologie en une nouvelle classe (C_{New}). Il nécessite l'application des règles de réécriture $AddClass(C_{New})$, $RemoveClass(C_1)$

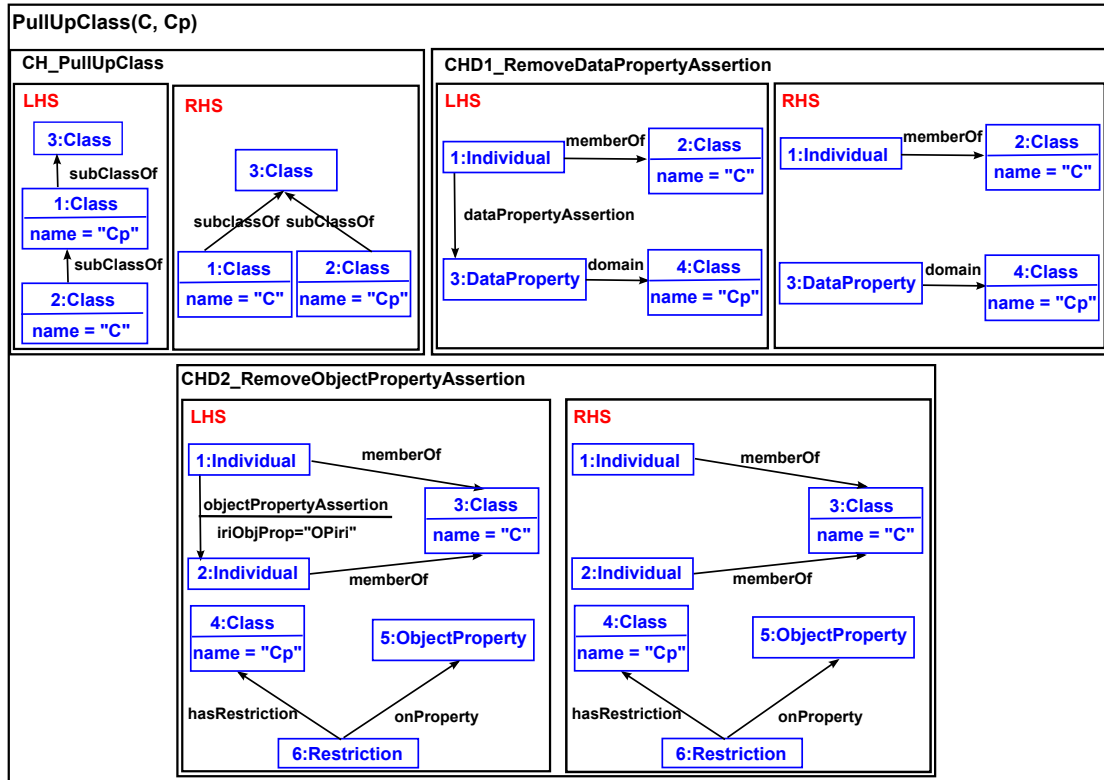


FIGURE 4.14 – Règle de réécriture du changement *PullUpClass*.

et *RemoveClass*(C_2). Cependant, pour préserver la consistance de l'ontologie, avant de supprimer C_1 et C_2 , toutes leurs propriétés et axiomes doivent être attachés à C_{New} . Formellement : 1) $\forall C_i \in C(O) \cdot C_i \sqsubseteq C_1$ appliquer la règle de réécriture *AddSubClass*(C_i, C_{New}) et $\forall C_j \in C(O) \cdot C_1 \sqsubseteq C_j$ appliquer *AddSubClass*(C_{New}, C_j), 2) répéter les mêmes règles de réécriture pour C_2 , 3) $\forall C_i \in C(O) \cdot C_i \equiv C_1$ appliquer *AddEquivalentClasses*(C_i, C_{New}), 4) répéter les mêmes règles de réécriture pour C_2 , etc.

4.5.3 Changements de division

Les changements de division concernent les classes et les propriétés. Nous présentons ici le changement de division appliquée sur une classe. Ainsi, le changement *SplitClass*(C, C_{New1}, C_{New2}) divise une classe (C) déjà existante dans l'ontologie en deux nouvelles classes C_{New1} et C_{New2} . Il nécessite l'application des règles de réécriture *AddClass*(C_{New1}), *AddClass*(C_{New2}) et *RemoveClass*(C). Comme le changement *MergeClasses*, le changement *SplitClass* nécessite, avant la suppression de la classe C , d'attacher toutes ses propriétés et axiomes aux classes

C_{New1} et C_{New2} .

4.6 Caractéristiques de l'approche proposée

Notre approche d'évolution d'ontologies EvOGG basée sur les grammaires de graphes possède deux principaux avantages :

1. fournir une nouvelle façon de formaliser des changements ontologiques permettant de mieux les contrôler et d'éviter les incohérences d'une manière a priori. Cette caractéristique permet de simplifier le processus d'évolution en fusionnant les étapes de 1) représentation de changement, 2) sémantique de changement et 3) propagation de changement en une seule étape qui est la formalisation de changement (voir Section 4.2.2). Ainsi, une règle de réécriture correspondante à un changement ontologique permet à la fois de définir le changement et de préserver la consistance de l'ontologie évoluée sans toutefois avoir recours à une ressource externe (les raisonneurs) pour vérifier la consistance de l'ontologie, ni retourner en arrière pour identifier les inconsistances et annuler les changements dont ils sont responsables.
2. simplifier la définition de certains changements composites et complexes tout en réduisant le nombre des règles de réécriture nécessaires pour les appliquer. Cette caractéristique permet de réviser la classification des changements (voir Figure 4.6). Nous pouvons classifier certains d'entre eux comme changements simples nécessitant qu'une seule règle de réécriture pour être formalisés.

Le Tableau 4.4 présente une comparaison des changements *AddObjectProperty*, *AddDataProperty* et *PullDownClass*, représentés à la fois par le formalisme proposé et le travail de [Djedidi and Aufaure, 2010] qui a été introduit dans le Chapitre 2, Section 2.4.3 et qui s'est intéressé également à la formalisation des changements ontologiques. Ainsi, dans Djedidi and Aufaure [2010], ces changements sont considérés comme composés et complexes. Le premier et le deuxième changement sont composés par trois changements élémentaires et le dernier changement est composé par deux. L'application de ces changements nécessite, d'ailleurs comme tous les autres changements, l'utilisation du raisonneur Pellet pour identifier d'une manière a posteriori les inconsistances. Dans notre travail, ces changements sont considérés comme élémentaires puisqu'ils ne sont composés que d'une seule règle de réécriture. De plus, pour préserver la consistance de l'ontologie, les inconsistances sont gérées d'une manière a priori grâce à l'utilisation des Negative Application Conditions (NAC). Dans les exemples présentés trois types d'inconsistances ont été traités : 1) redondance de données, 2) concepts isolés et 3) axiomes contradictoires.

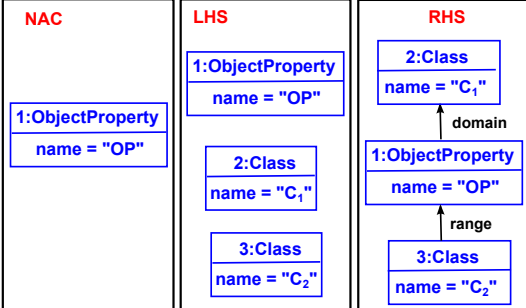
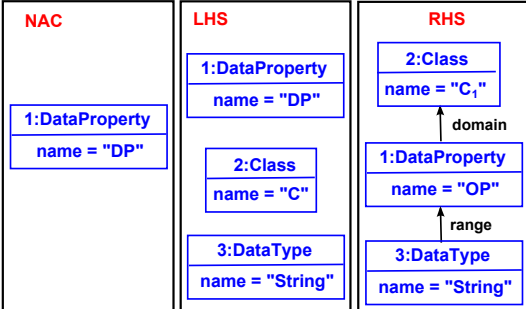
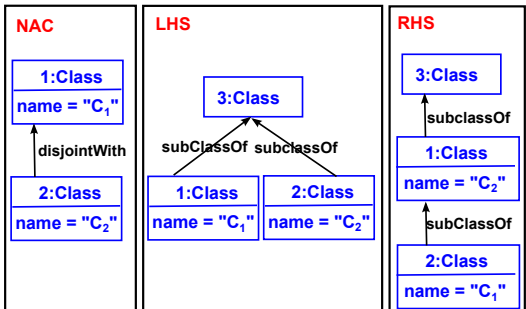
Changement ontologique	[Djedidi and Aupaure, 2010]	Formalisme proposé
<p><i>AddObjectProperty</i>(OP, C_1, C_2)</p>	<p>Le changement est composé de trois changements élémentaires :</p> <ol style="list-style-type: none"> 1. <i>AddObjectProperty</i>(OP), 2. <i>AddDomain</i>(OP, C_1) 3. <i>AddRange</i>(OP, C_2). 	<p>- Le changement est formalisé par une seule règle de réécriture et évite la redondance de données l'existence d'un concept isolé.</p> 
<p><i>AddDataProperty</i>(DP, C, DT)</p>	<p>Le changement est composé de trois changements élémentaires :</p> <ol style="list-style-type: none"> 1. <i>AddDataProperty</i>(DP), 2. <i>AddDomain</i>(DP, C) 3. <i>AddRange</i>(DT, DP). 	<p>- Le changement est formalisé par une seule règle de réécriture et évite la redondance de données et l'existence d'un concept isolé.</p> 
<p><i>PullDownClass</i>(C_1, C_2) : descend une classe (C_1) de sa hiérarchie de classes et l'attacher comme sub-classe de sa précédente classe sœur (C_2).</p>	<p>Le changement est composé par deux changements élémentaires :</p> <ol style="list-style-type: none"> 1. <i>AddSubClass</i>(C_1, C_2) 2. <i>RemoveSubClass</i>(C_1, C_p) avec la classe C_p est la super-classe de C_1 et C_2. 	<p>- Le changement est formalisé par une seule règle de réécriture et évite la contradiction des axiomes.</p> 

TABLE 4.4: Formalisation des changements ontologiques selon [Djedidi and Aupaure, 2010] et l'approche proposée.

4.7 Bilan

Dans ce chapitre, nous avons présenté une approche d'évolution d'ontologies basée sur les grammaires de graphes et les transformations algébriques de graphes. Nous avons décrit notre modèle de représentation d'ontologies et leurs changements ontologiques, $TGG_{Onto} = \{GT_O, G_O, R_O\}$ qui permet de représenter : 1) une ontologie sous forme d'un graphe attribué et étiqueté, 2) le méta-modèle du langage OWL comme un graphe type et 3) les changements ontologiques comme des règles de réécritures implémentées par l'approche algébrique Simple Pushout (SPO). Une formalisation détaillée sur les différents types de changements (élémentaires, composés et complexes) a été également présentée permettant de vérifier la consistance de l'ontologie évoluée d'une manière a priori.

Nous présentons dans le chapitre suivant comment utiliser le formalisme défini pour formaliser le processus de fusion d'ontologies.

Contributions et valorisation

Les recherches présentées dans ce chapitre ont permis de mieux comprendre les différents changements que peut subir une ontologie (leurs types, leurs niveaux, leurs pre-conditions d'exécutions, leurs post-conditions, etc). Un couplage entre les ontologies et les grammaires de graphe a été proposé, une nouvelle approche d'évolution d'ontologies basée sur les grammaires de graphes a été définie et une gestion des inconsistances d'une manière a priori a été établie.

Les travaux effectués dans ce chapitre ont pu être validés et valorisés à travers quatre publications (trois communications et un article dans une revue internationale). La première communication [Mahfoudh et al., 2013b] était dans une conférence nationale spécialisée dans l'ingénierie et la gestion des connaissances (*Ingénierie des Connaissances*). Elle a introduit notre formalisme de changements ontologiques basé sur les grammaires de graphes en insistant sur la possibilité de couplage entre les ontologies et les grammaires de graphes. La deuxième communication [Mahfoudh et al., 2013a] était dans une conférence internationale (*Knowledge Science, Engineering and Management*). Elle a présenté la formalisation des changements élémentaires avec le formalisme proposé. La troisième [Mahfoudh et al., 2015b] était dans une conférence nationale (*Extraction et Gestion des Connaissances*). Elle s'est focalisée sur la formalisation des changements composés et complexes. Enfin, l'article [Mahfoudh et al., 2015a] a été publié dans un journal international (*Knowledge-Based Systems*). Il englobe et détaille tous les types des changements étudiés. Il présente, de plus, le cadre applicatif de notre travail : outils développés et les ontologies définies.

Chapitre 5

Approche de fusion d'ontologies

Sommaire

5.1 Introduction	107
5.2 Approche de fusion d'ontologies	108
5.2.1 Exemples introductifs	108
5.2.2 Processus global de fusion d'ontologies	110
5.3 Recherche de similarité	112
5.4 Fusion des ontologies	113
5.4.1 Appariement d'ontologies	114
5.4.2 Construction de l'ontologie commune	115
5.4.3 Construction de l'ontologie globale	116
5.5 Adaptation de l'ontologie globale	116
5.5.1 Enrichissement par des relations de synonymies	117
5.5.2 Enrichissement par des relations de subsumptions	117
5.6 Bilan	119

"Cheville ! redondance inutile !"

Jean-Jacques Rousseau

5.1 Introduction

La prolifération des ontologies a conduit au développement de ressources complémentaires ou partiellement redondantes modélisant des domaines identiques ou connexes. Cette multiplication de ressources disponibles a mené aux études traitant leur réutilisation et/ou leur fusion. Dans ce cadre, la fusion d'ontologies consiste à fusionner deux ontologies ou plus pour créer une nouvelle ontologie, souvent appelée ontologie globale.

Ainsi, le problème de fusion consiste dans un premier temps à trouver l'alignement entre des ontologies et ceci en cherchant les correspondances entre leurs concepts. Puis, dans un

second temps, il s'agit de trouver l'union en se basant sur l'alignement défini. Pour accomplir la première phase, plusieurs techniques de similarité ont été proposées dans la littérature (voir Chapitre 2, Section 2.5). Il s'agit d'un axe de recherche largement étudié qui présente des résultats de plus en plus importants [Pavel and Euzenat, 2013]. Quant à la phase de fusion d'ontologies, peu de travaux existent et plusieurs questions restent en suspens. Comment fusionner des ontologies ? Quel cadre formel peut-on utiliser pour suivre ce processus ? Quelle est la "meilleure" stratégie de fusion "symétrique" ou bien "asymétrique" ? etc.

Afin de répondre à ces questions, nous proposons une approche formelle de fusion d'ontologies appelée GROM (Graph Rewriting for Ontologies Merging) qui se base sur les règles de réécriture de graphes. Nous faisons ainsi appel à notre formalisme $TGG_{Onto} = \{GT_O, G_O, R_O\}$ introduit dans le chapitre précédent pour représenter les ontologies et réaliser les différentes transformations nécessaires au processus de fusion. L'approche proposée est asymétrique et se décompose en trois grandes étapes : (1) la recherche de similarité entre concepts d'ontologies, (2) la fusion par l'approche algébrique Simple PushOut (SPO) et (3) l'adaptation de l'ontologie globale par les règles de réécriture.

L'objectif de ce chapitre est de présenter notre approche de fusion d'ontologies basée sur les grammaires de graphes. Nous commençons par introduire la problématique de fusion en montrant des exemples illustratifs d'ontologies et des conflits pouvant résulter de leur fusion. Ensuite, nous décrivons l'approche proposée et nous détaillons ses différentes étapes.

5.2 Approche de fusion d'ontologies

5.2.1 Exemples introductifs

Afin de mieux comprendre le processus général de fusion d'ontologies et ses enjeux, nous présentons dans ce qui suit deux exemples de fusion d'ontologies. Le premier exemple concerne des taxonomies (une hiérarchie des concepts liés avec des relations de subsomption). Le deuxième exemple présente des ontologies plus expressives définies par le langage OWL (voir Chapitre 2, Section 2.2.4).

Exemple 1 : cas des taxonomies

La Figure 5.1 présente un exemple de deux ontologies sous forme de taxonomies. L'ontologie O_1 se décompose en six classes et la deuxième ontologie O_2 se décompose en sept classes. Les deux ontologies modélisent un même domaine d'étude qui est le domaine d'automobile. Ces ontologies partagent des concepts communs ("Automobile", "BMV", "Fiat") et des relations de subsomption (isA ("German_Car", "European_Car"), isA ("Italian_Car", "European_Car") et

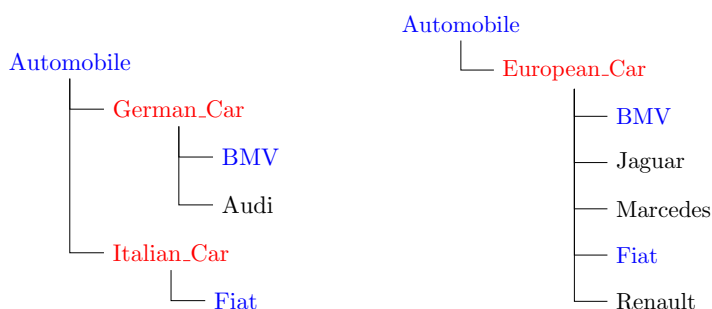


FIGURE 5.1 – Exemple de deux taxonomies.

isA ("Mercedes", "German_Car"). La fusion de deux taxonomies peut engendrer les situations et les conflits suivants :

- *Redondance de données*. Étant donné que les ontologies partagent des concepts communs, leur résultat de fusion peut contenir des éléments redondants, par exemple ("Automobile", "Automobile").
- *Partage des relations de subsomption*. Les ontologies peuvent partager des relations de subsomption, par exemple isA("German_Car", "European_Car") et isA("Italian_Car", "European_Car").
- *Existence de cycle*. L'ajout des relations de subsomption peut engendrer des cycles. Par exemple, si nous fusionnons les deux ontologies de la Figure 5.1 et nous ajoutons les relations de subsomptions qu'elles partagent, nous aurons le cycle suivant : isA("German_Car", "European_Car"), isA("German_Car", "Automobile"), isA("European_Car", "Automobile").
- *Axiomes contradictoires*. Les taxonomies peuvent contenir des axiomes de subsomptions contradictoires.

Exemple 2 : cas des ontologies OWL

Nous reprenons l'exemple précédent et nous les enrichissons par des concepts et des axiomes du langage OWL (Figure 5.2.1). Les deux ontologies de ce nouvel exemple présentent, de plus, des individus, des propriétés (ObjectProperty), des restrictions (CardinalityRestriction) et des axiomes (domain, range, etc.). La fusion des ontologies OWL peut provoquer de plus les situations et les conflits suivants :

- *Redondance de données*. Les concepts qui ont des termes syntaxiquement proches peuvent être considérés dans l'ontologie globale comme de l'information redondante, par exemple "has_owner" et "hasOwner".

- *Concepts synonymes*. Les ontologies peuvent partager des relations de synonymies, par exemple, "Individual" et "Person".
- *Axiomes contradictoires*. La fusion de deux ontologies OWL peut engendrer différents types d'axiomes contradictoires qui touchent la disjonction, l'équivalence, les restrictions, etc.

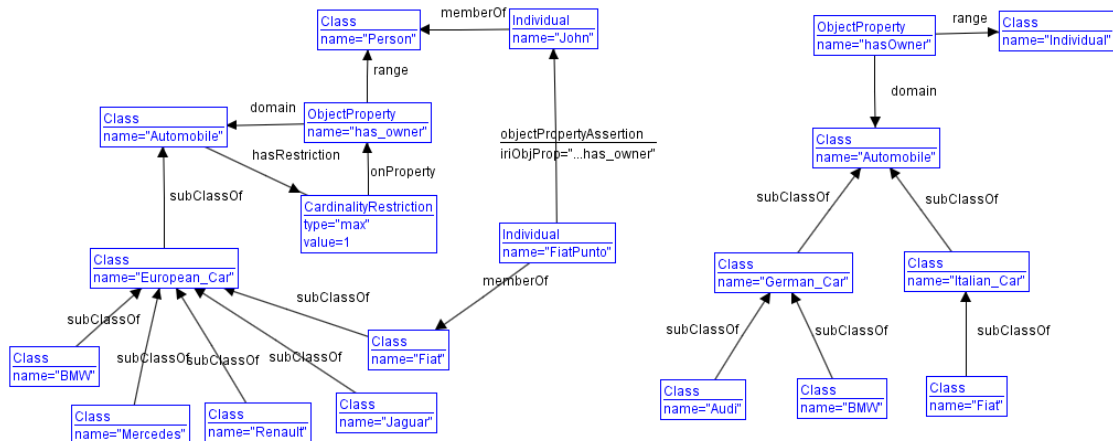


FIGURE 5.2 – Exemple de deux ontologies OWL.

Afin de traiter les conflits et les situations présentés ci-dessus, nous proposons d'utiliser le formalisme de grammaires de graphes.

5.2.2 Processus global de fusion d'ontologies

Nous proposons une approche de fusion d'ontologies GROM (Graph Rewriting for Ontologies Merging) composée de trois grandes étapes (Figure 5.3) :

1. La *recherche de similarité* qui permet de déterminer les correspondances entre les concepts des ontologies à fusionner. Nous nous sommes intéressés dans ce travail à identifier trois types de similarités : syntaxique, structurelle et sémantique (voir Section 5.3).
2. La *fusion des ontologies* qui permet de fusionner les ontologies en se basant sur les correspondances trouvées dans l'étape précédente. Cette étape nécessite tout d'abord de représenter les ontologies avec le formalisme des grammaires de graphes typés adapté aux ontologies $TGG_{Onto} = \{GT_O, G_O, R_O\}$. Ensuite, elle permet de fusionner les ontologies à l'aide des règles de réécriture et l'approche algébrique SPO. Elle se décompose de trois sous étapes : (1) l'appariement d'ontologies, (2) la création de l'ontologie com-

mune et (3) la construction de l'ontologie globale. Plus de détails sont donnés dans la Section 5.4.

- 3. l'adaptation de l'ontologie globale qui permet d'enrichir l'ontologie résultant de l'étape précédente par les relations de subsomption et de synonymies identifiées dans la première étape. Plus de détails sont donnés dans la Section 5.5.

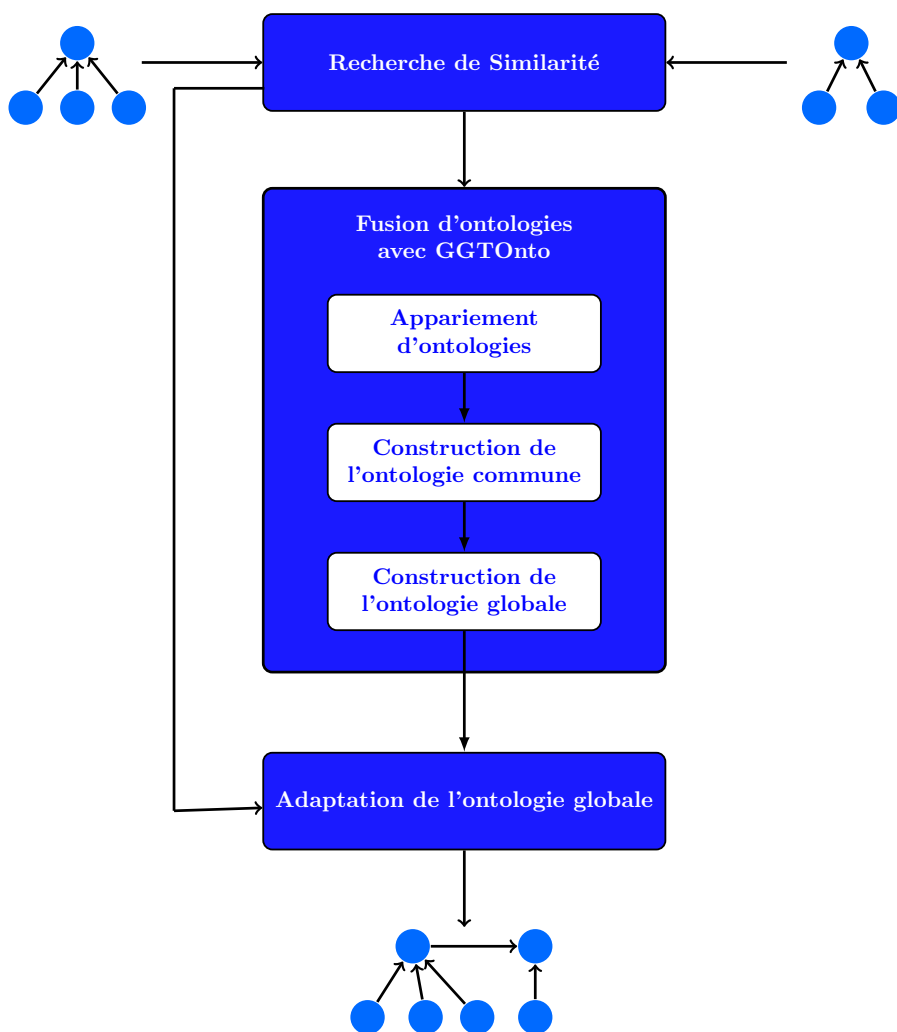


FIGURE 5.3 – Approche GROM de fusion d'ontologies.

5.3 Recherche de similarité

Dans notre travail, nous nous intéressons à trois types de similarité :

1. la similarité syntaxique qui permet d'identifier la correspondance syntaxique entre les concepts des ontologies. Nous utilisons ce type de similarité pour déterminer l'ensemble des nœuds communs et syntaxiquement équivalents (ou peu différent) entre les ontologies à fusionner ;
2. la similarité sémantique qui permet d'identifier les relations sémantiques entre les concepts des ontologies. Nous nous sommes intéressés plus particulièrement à la relation de synonymie (voir Chapitre 2, Section 2.1). Ce type de similarité permet d'enrichir le résultat de fusion et favorise sa réutilisation dans le domaine de recherche d'information et la reformulation des requêtes [Mahfoudh and Jaziri, 2013] ;
3. la similarité structurelle qui permet d'identifier les relations de subsomption entre les concepts des ontologies. Nous l'utilisons aussi pour enrichir l'ontologie globale.

En se basant sur ces différents types de similarité, nous identifions, ainsi, quatre types d'ensemble de nœuds :

1. NC est l'ensemble des nœuds communs entre les deux ontologies possédant des noms identiques. Formellement, $NC = \{N_i | (N_i \in N(O_1)) \wedge (\exists N_j \in N(O_2) \cdot (N_{iT} = N_{jT}) \wedge (distanceSyntaxique(N_i.name, N_j.name) = 0))\}$. Cette identification vise à éviter les nœuds redondants dans le résultat final de la fusion. Sur l'exemple des ontologies illustrées dans la Figure 5.2.1, l'ensemble des nœuds communs est $NC = \{ "Automobile", "Fiat", "BMW" \}$.
2. NE est l'ensemble des nœuds équivalents dont les chaînes de caractères de leurs noms sont syntaxiquement proches. Formellement, $NE = \{(N_i, N_j) | (N_i \in N(O_1)) \wedge (N_j \in N(O_2)) \wedge (N_{iT} = N_{jT}) \wedge (distanceSyntaxique(N_i.name, N_j.name) < seuil)\}$. Par exemple, $NE = \{ ("has_owner", "hasOwner") \}$. Dans notre travail, nous considérons que ces concepts sont équivalents et seule un d'entre eux doit être gardé dans l'ontologie globale.
3. Nœuds partageant une relation sémantique de synonymie, $NS = \{(N_i, N_j) | (N_i \in N(O_1)) \wedge (N_j \in N(O_2)) \wedge (N_{iT} = N_{jT}) \wedge (distanceSémantique(N_i.name, N_j.name) = 0)\}$. Par exemple, $NS = \{ ("Person", "Individual") \}$
4. Nœuds partageant une relation de subsomption, $NIsa = \{(N_i, N_j) | (N_i \in N(O_1)) \wedge (N_j \in N(O_2)) \wedge (N_{iT} = N_{jT}) \wedge (Isa(N_i) = N_j)\}$. Par exemple, $NIsa = \{ ("German_Car", "European_Car"), ("Italian_Car", "European_Car"), ("Mercedes", "German_Car") \}$.

5.4 Fusion des ontologies

L'étape de fusion d'ontologies consiste à créer une ontologie globale représentant le résultat de leur union. Pour réaliser ceci, deux types d'approches sont généralement distinguées dans la littérature : 1) les approches symétriques (*full merge*) qui permettent de fusionner des ontologies tout en préservant la totalité de leur contenu dans l'ontologie globale ; 2) les approches asymétriques (*source-driven merge* et *target-driven merge*) qui considèrent l'une des deux ontologies comme source et l'autre comme destination (plus de détails dans le Chapitre 2, Section 2.5.2).

Dans notre travail, nous nous plaçons dans le cadre des approches asymétriques. Soient deux ontologies O_1 et O_2 , alors $Merge(O_1, O_2) \neq Merge(O_2, O_1)$. Les concepts de l'ontologie source seront préservés alors que seuls les concepts non redondants de l'ontologie destination, et qui n'affectent pas la consistance de l'ontologie source, seront ajoutés à l'ontologie globale. En effet, dans les approches asymétriques le choix de l'ontologie source est considéré comme important puisque ses concepts seront tous préservés. En revanche, certains concepts de l'ontologie destination seront supprimés. En général, le choix entre les deux ontologies peut être déterminé manuellement ou bien semi-automatique. Dans ce dernier cas, il faut comparer la qualité des deux ontologies en se basant sur différents critères : (1) la consistance (si une ontologie est inconsistante alors l'autre sera la source) ; (2) la taille (l'ontologie la plus grande en terme de nombre de nœuds est considérée comme source) ; (3) la profondeur (favoriser l'ontologie la plus profonde en terme de niveau de hiérarchie) ; (4) l'expressivité (l'ontologie la plus expressive contenant plus d'axiomes est considérée comme l'ontologie source) ; etc. Actuellement, nous ne nous sommes pas intéressés à ce type de comparaison. Nous considérons que les ontologies à fusionner sont consistantes et que c'est à l'utilisateur de préciser l'ontologie source. Ce choix ne présente pas un grand défi pour l'utilisateur puisque généralement dans les systèmes d'information, il y a souvent une ontologie noyau et les autres ontologies sont ajoutées pour l'enrichir.

Les approches asymétriques sont beaucoup plus efficaces pour la gestion des conflits. Elles permettent de n'ajouter que les éléments cohérents ce qui est en parfaite adéquation avec notre approche d'évolution d'ontologies de résolution a priori des inconsistances (voir Chapitre 4). Ainsi, nous reprenons le formalisme des grammaires de graphes défini dans la chapitre précédent, $TGGOnto = \{GT_O, G_O, R_O\}$. Les ontologies sont alors représentées par des graphes hôtes (des objets de la catégorie *Graph*) et les transformations sont réalisées par l'approche algébrique Simple Pushout (SPO). Le processus de fusion d'ontologies repose sur le résultat de l'alignement trouvé dans l'étape précédente et il est décrit par l'algorithme 1. Il consiste tout d'abord à appairer les deux ontologies. Ensuite, il faut construire l'ontologie commune pour

les fusionner et enfin, adapter l'ontologie globale résultante. Nous détaillons toutes ces étapes dans les sections suivantes.

Algorithme 1 : Algorithme de fusion d'ontologies

Entrées : deux ontologies O_1, O_2
 un ensemble de correspondances : $NC, NE, NS, NIsa$

Sorties : une ontologie globale OG

pour $N \in NE$ **faire**
 ┌ $O'_1 \leftarrow SPO_RenameEntity (O_1, NE\{O_1\}, NE\{O_2\})$;
 $NC \leftarrow NC \cup NE\{O_1\} ;$
 $OC \leftarrow$ Créer le graphe de l'ontologie commune ;
 $OG \leftarrow SPO_MergeGraph (O'_1, OC, O_2)$;

/ Adapter l'ontologie globale */*

pour $N \in NS$ **faire**
 ┌ $OG \leftarrow SPO_AddEquivalentEntity (OG, NS\{O_1\}, NS\{O_2\})$;

pour $N \in NIsa$ **faire**
 ┌ $OG \leftarrow SPO_AddSubClass (OG, NIsa\{O_1\}, NIsa\{O_2\})$;

5.4.1 Appariement d'ontologies

L'étape d'appariement d'ontologies vise à minimiser la différence entre les deux ontologies à fusionner et préparer par la suite la création de l'ontologie commune qui va éviter les redondances. Son rôle consiste à remplacer les entités de l'ontologie O_1 par leurs équivalents de l'ontologie O_2 . Elle se base ainsi sur l'ensemble des nœuds équivalents (NE) et consiste à appliquer un ensemble de règles de réécriture avec l'approche SPO (voir Figure 5.4).

Les règles de réécriture correspondent aux changements de renommage. Il s'agit d'appliquer des règles de *RenameEntity* (N_i, N_j) avec N_i est un nœud de O_1 et N_j est son équivalent dans O_2 . Ainsi, des SPO seront appliqués pour tout l'ensemble de NE et ils sont définis par :

- le graphe hôte est l'ontologie O_1 ;
- le *LHS* est le graphe constitué de l'ensemble des nœuds $\{N_i \in NE\}$;
- le *RHS* est le graphe constitué de l'ensemble des nœuds $\{N_j \in NE\}$.

Les entités à renommer peuvent être de type classe, propriété ou bien individu. Comme exemple, la Figure 5.5 présente la règle de réécriture du changement *RenameObjectProperty*.

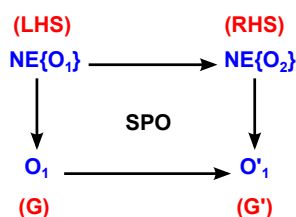


FIGURE 5.4 – Appariement d'ontologies avec SPO.

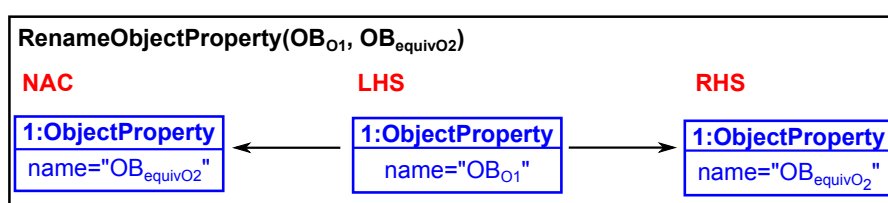


FIGURE 5.5 – Règle de réécriture du changement RenameObjectProperty.

5.4.2 Construction de l'ontologie commune

L'étape 2 de fusion d'ontologies consiste à créer l'ontologie $OC = (N, E)$ qui est le sous-graphe commun entre les deux ontologies O'_1 (la nouvelle version de l'ontologie O_1 obtenue après la phase de l'appariement) et O_2 (Figure 5.6). Formellement, il s'agit d'un span (opérateur introduit dans le Chapitre 3, Section 3.3.1). Ainsi, l'ensemble des nœuds de l'ontologie commune, $N = NC \cup NE\{O_2\}$, correspond à l'union entre les nœuds communs (NC) identifiés dans la phase de la recherche de similarité et les nœuds $NE\{O_2\}$. Les arêtes, E , sont construits en identifiant les arêtes liant les nœuds N dans l'ontologie O'_1 et qui ne sont pas en contradiction avec ceux de O_2 . En effet, les arêtes représentent les axiomes d'ontologies (disjointWith , subClassOf , etc.). De là, l'ontologie commune ne doit pas représentée les axiomes contradictoires entre les deux ontologies (sources et destination) pour ne pas altérer le résultat de fusion.

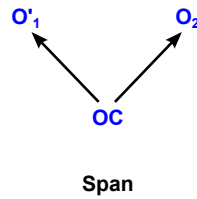


FIGURE 5.6 – Construction de l'ontologie commune.

5.4.3 Construction de l'ontologie globale

La construction de l'ontologie globale est réalisée par la règle de réécriture *MergeGraph*. Cette règle de production a comme graphe hôte l'ontologie O'_1 (O_1 après modification). Son *LHS* est l'ontologie commune OC et son *RHS* est l'ontologie O_2 (voir la Figure 5.7). Elle a pour rôle l'intégration des deux ontologies et ceci en les liant par leurs entités communes. Le résultat de cette transformation donne une première version de l'ontologie globale (OG) qui sera encore modifiée et enrichie par l'ajout des relations sémantiques et de subsomption.

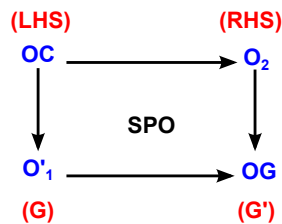


FIGURE 5.7 – Règle de réécriture de la construction de l'ontologie globale.

5.5 Adaptation de l'ontologie globale

Étant donné que la transformation de graphe nécessite la présence d'un *matcher* (morphisme m) entre le *LHS* et le graphe hôte, l'ajout des relations de subsomption et de relations sémantiques doit être appliqué après la création de l'ontologie globale. Par conséquent, il faut appliquer les règles de réécriture *AddEquivalentEntity* et *AddSubClass* qui ont pour rôle d'enrichir l'ontologie globale sans affecter sa consistance. La vérification des inconsistances est assurée par les conditions négatives d'application (*NAC*).

5.5.1 Enrichissement par des relations de synonymies

La règle de réécriture *AddEquivalentEntity* ajoute un axiome d'équivalence entre deux entités : deux classes ou deux propriétés. La Figure 5.8 présente l'exemple de la règle de réécriture de *AddEquivalentClasses* (C_1, C_2) qui est défini comme suit :

- NACs :
 1. $C_1 \equiv C_2$, condition pour éviter la redondance ;
 2. $C_1 \sqsubseteq \neg C_2$, deux classes ne peuvent pas être à la fois disjointes et équivalentes ;
- LHS : $\{C_1, C_2\}$, les classes doivent exister dans l'ontologie.
- RHS : $(C_1 \equiv C_2)$, l'axiome doit être ajouté à l'ontologie.

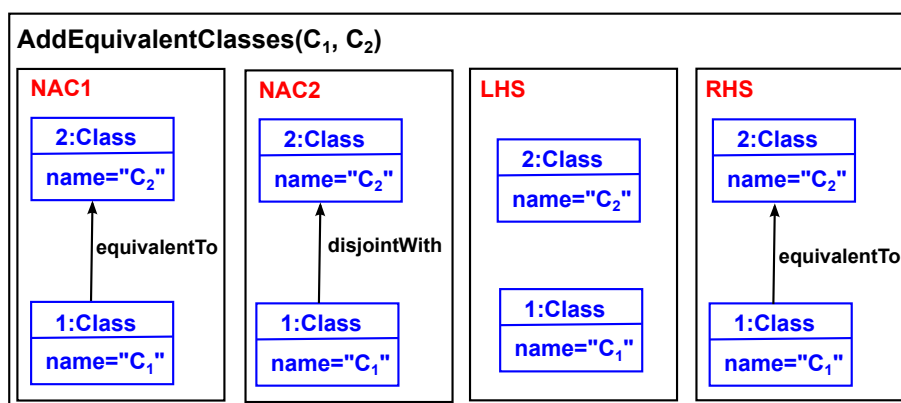


FIGURE 5.8 – Règle de réécriture du changement *AddEquivalentClasses*.

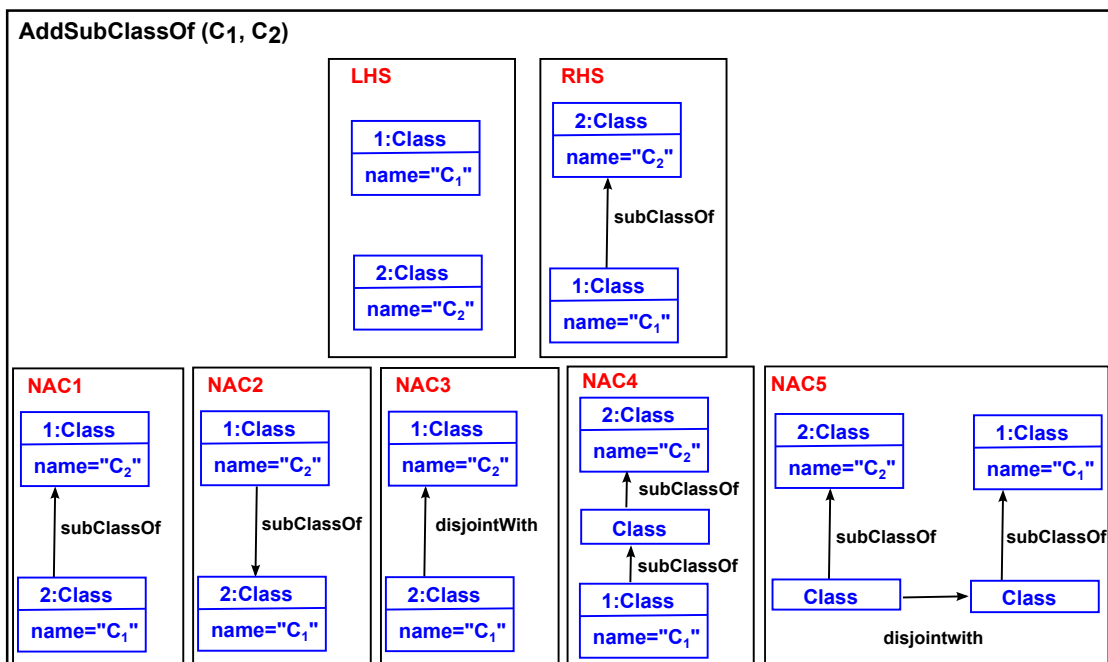
5.5.2 Enrichissement par des relations de subsumptions

A son tour, la règle de réécriture *AddSubClass* (Figure 5.9) ajoute l'axiome *subClassOf* entre deux classes et elle est définie comme suit :

- NACs :
 1. $C_1 \sqsubseteq C_2$, condition pour éviter la redondance ;
 2. $C_2 \sqsubseteq C_1$, la relation de subsumption ne peut pas être symétrique ;
 3. $C_1 \sqsubseteq \neg C_2$, les classes qui partagent une relation de subsumption ne peuvent pas être disjointes ;

4. $\exists C_i \in C(O) \cdot (C_1 \sqsubseteq C_i) \wedge (C_i \sqsubseteq C_2)$. S'il existe dans l'ontologie une classe C_i qui est à la fois *subClass* de la classe C_2 et *superClass* de C_1 , alors, C_1 est déjà la *subClass* de C_2 et il ne faut pas ajouter ce lien de subsomption ;
5. $\exists (C_i, C_j) \in C(O) \cdot (C_i \sqsubseteq C_1) \wedge (C_j \sqsubseteq C_2) \wedge C_i \sqsubseteq \neg C_j$, les classes qui partagent une relation de subsomption ne peuvent pas avoir des subClasses disjointes.

- LHS : $\{C_1, C_2\}$, les classes doivent exister dans l'ontologie.
- RHS : $(C_1 \sqsubseteq C_2)$, l'axiome doit être ajouté à l'ontologie.

FIGURE 5.9 – Règle de réécriture du changement *AddSubClass*.

Si nous prenons l'exemple des ontologies de la Figure 5.2.1, le résultat de leur fusion selon l'approche proposée GROM sera celui illustré par la Figure 5.10. Nous pouvons constater que les concepts redondants sont supprimés ("Automobile", "Fiat", "BMW", "has_Owner"). Les relations de subsomption sont ajoutées tout en supprimant le cycle entre elles (les relations $\text{isA}(\text{"German_Car"}, \text{"Automobile"})$ et $\text{isA}(\text{"Italian_Car"}, \text{"Automobile"})$ sont supprimées puisqu'elles engendrent des cycles). Enfin, les relations de synonymies sont ajoutées ("Person" est équivalent à "Individual").

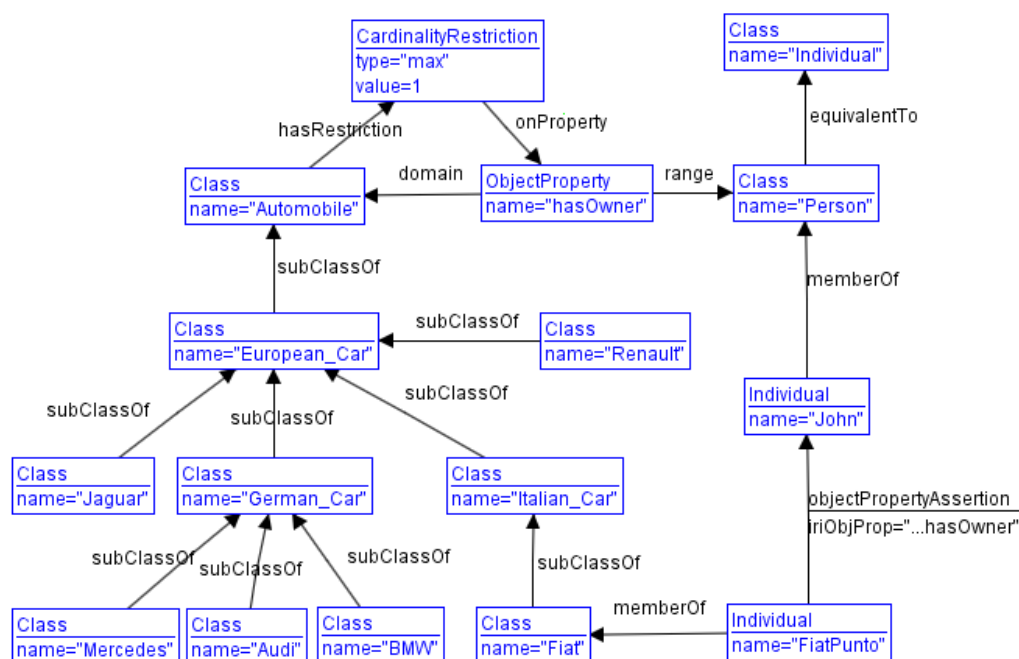


FIGURE 5.10 – Résultat de fusion des ontologies illustrées dans la Figure 5.2.1 par l’approche GROM.

5.6 Bilan

Nous avons présenté dans ce chapitre une méthode de fusion d’ontologies basée sur la réécriture de graphes. L’idée principale est de profiter du formalisme défini pour l’évolution d’ontologies et de l’appliquer à la problématique de fusion, celle-ci pouvant aussi être considérée comme une forme de changement d’ontologies.

L’approche proposée est une approche asymétrique permettant d’éviter la redondance dans l’ontologie globale et de réduire les conflits pouvant résulter du processus de fusion. Cette résolution est assurée par les règles de réécriture et les conditions négatives d’applications (NAC).

L’utilisation de l’approche algébrique SPO qui encapsule les détails complexes de la structure des ontologies en les considérant comme des objets d’une catégorie (*Graph*) a permis de simplifier la recherche des patterns et l’intégration des graphes.

Contributions et valorisation

Les travaux effectués dans ce chapitre ont pu être validés et valorisés à travers trois communications. La première [Mahfoudh et al., 2014a] dans une conférence nationale (*Extraction et Gestion des Connaissances*). La deuxième [Mahfoudh et al., 2014c] dans une conférence internationale (*International Conference on Model & Data Engineering*) et la troisième [Mahfoudh et al., 2014b] dans une conférence francophone (*Journées Francophones sur les Ontologies*).

Chapitre 6

Application aux ontologies CCAIps

Sommaire

6.1 Introduction	121
6.2 Cadre applicatif du travail	122
6.2.1 Ontologies CCAIps	122
6.2.2 Changements ontologiques dans les ontologies CCAIps	125
6.3 Implémentation	128
6.3.1 Les outils OWLToGGX et GGXToOWL	128
6.3.2 Approche EvOGG	131
6.3.3 Approche GROM	132
6.4 Résultats et évaluation	133
6.4.1 Transformation d'ontologies OWL en graphes hôtes	133
6.4.2 Approche d'évolution	133
6.4.3 Approche de fusion	136
6.5 Bilan	139

"La théorie, c'est quand on sait tout et que rien ne fonctionne. La pratique, c'est quand tout fonctionne et que personne ne sait pourquoi. Ici, nous avons réuni théorie et pratique : rien ne fonctionne... et personne ne sait pourquoi !"

Albert Einstein

6.1 Introduction

Dans les chapitres précédents, nous avons étudié la problématique de la réutilisation d'ontologies en proposant un cadre formel pour décrire leur évolution et leurs fusion. Plus précisément, nous avons proposé l'utilisation des grammaires de graphes typés pour définir deux approches. La première approche EvOGG (Evolving Ontologies with Graph Grammars) définit

une formalisation de l'évolution d'ontologies avec une résolution a priori des inconsistances. La deuxième approche GROM (Graph Rewriting for Ontologies Merging) présente une approche asymétrique de fusion d'ontologies. Afin d'implémenter et de valider ces propositions, nous avons développé des outils en langage Java basés sur l'outil AGG (Attributed Graph Grammar). Comme exemple d'application, nous avons appliqué les approches proposées sur un ensemble d'ontologies, entre autres les ontologies développées dans le cadre du projet European CCAIps.

Ce chapitre présente les outils développés tout au long de cette thèse. Nous présentons en premier lieu le cadre applicatif de notre travail. Nous introduisons ainsi le projet CCAIps, ses ontologies et certains changements identifiés. Ensuite, nous introduisons les outils implémentés : OWLToGGX, GGXToOWL, EvOGG et GROM. Enfin, nous présentons quelques résultats obtenus.

6.2 Cadre applicatif du travail

Ce travail de cette thèse s'inscrit dans le cadre du projet européen CCAIps¹ (Creative Companies in Alpine Space). Le projet consistait à proposer une plateforme collaborative devant aider à mettre en relation les entreprises créatives et les régions de l'Espace Alpin (numéro de projet 15-3-1-IT). Six partenaires de différents pays (la France, l'Allemagne, l'Italie, la Suisse, la Slovénie et l'Autriche) participent au projet. Les différents partenaires du projet organisent régulièrement des événements et ont besoin de représenter et gérer des informations qui sont constamment modifiées et changées. Ainsi, afin de représenter les informations et dans le but de créer un consensus entre les différents acteurs du projet, l'idée a été de faire appel aux ontologies.

Nous présentons dans cette section les ontologies développées dans le cadre du projet et la liste de leurs changements les plus importants.

6.2.1 Ontologies CCAIps

Durant son cycle de vie, le projet CCAIps a passé par plusieurs phases. En effet, les besoins des différents partenaires ont régulièrement changés et à chaque fois il a fallu adapter les données modélisées. Au début du projet, quatre ontologies OWL ont été développées : RegionCCAIps, EventCCAIps, CompanyCCAIps et HubCCAIps. Une vue d'ensemble des relations entre ces ontologies est donnée sur la Figure 6.1. Nous décrivons ci-après seulement les trois

1. www.ccalps.eu



premières ontologies puisque l'ontologie HubCCAIPs a été abandonnée avec l'évolution du projet.

Les ontologies sont présentées ici en utilisant l'utilitaire *Mot+onto*² (un éditeur graphique d'ontologies OWL).

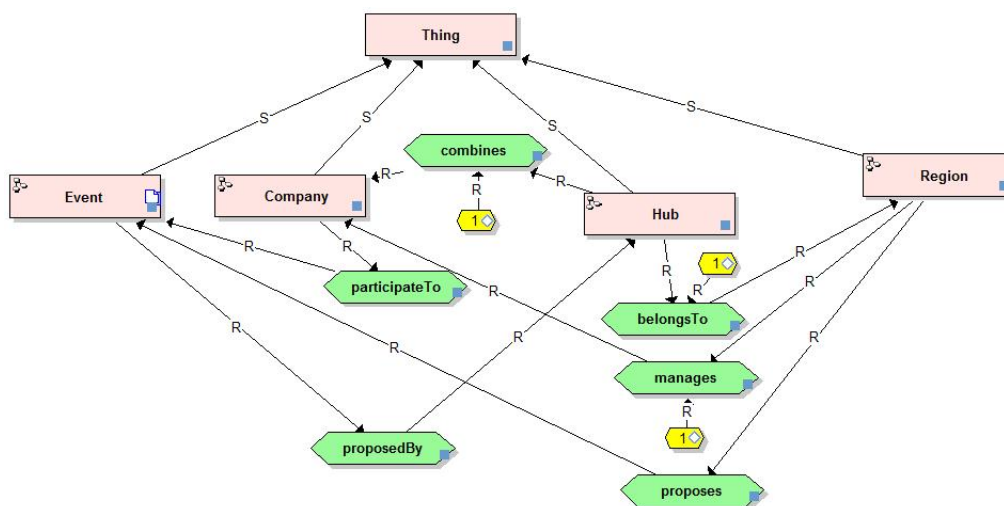


FIGURE 6.1 – Relation entre les ontologies CCAIps.

L'ontologie RegionCCAIPs

L'ontologie RegionCCAIPs définit les régions participant au projet. Elle présente les caractéristiques d'une région (sa description, sa localisation, etc.). Une région peut être créée et administrée par une personne enregistrée dans la plateforme. Elle est également décrite par un ensemble de "tags". En relation avec les autres ontologies, une région gère un ensemble d'entreprises (relation par rapport à l'ontologie CompanyCCAIPs) et propose un ensemble d'évènements (relation par rapport à l'ontologie EventCCAIPs). La Figure 6.2 présente un extrait de l'ontologie RegionCCAIPs.

2. cogigraph.com/Produits/MOTetMOTplus/tabid/995/language/fr-FR/Default.aspx

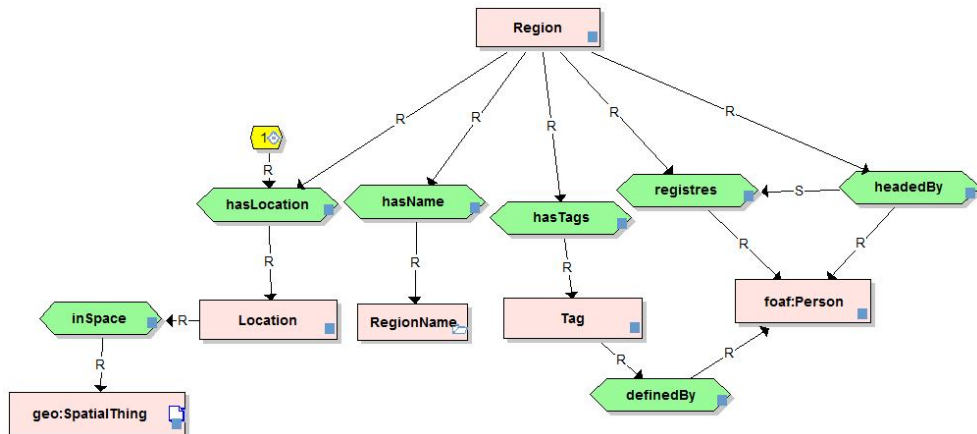


FIGURE 6.2 – Extrait de l'ontologie RegionCCAlps.

L'ontologie EventCCAlps

L'ontologie EventCCAlps définit les concepts liés aux événements organisés dans le cadre du projet. Elle représente les caractéristiques d'un événement (sa description, sa localisation, sa période, etc.). En relation avec les autres ontologies, un événement est créé par une région et regroupe un ensemble de participants, qui peuvent être des particuliers ou bien aussi des entreprises. La Figure 6.3 présente un extrait de l'ontologie EventCCAlps.

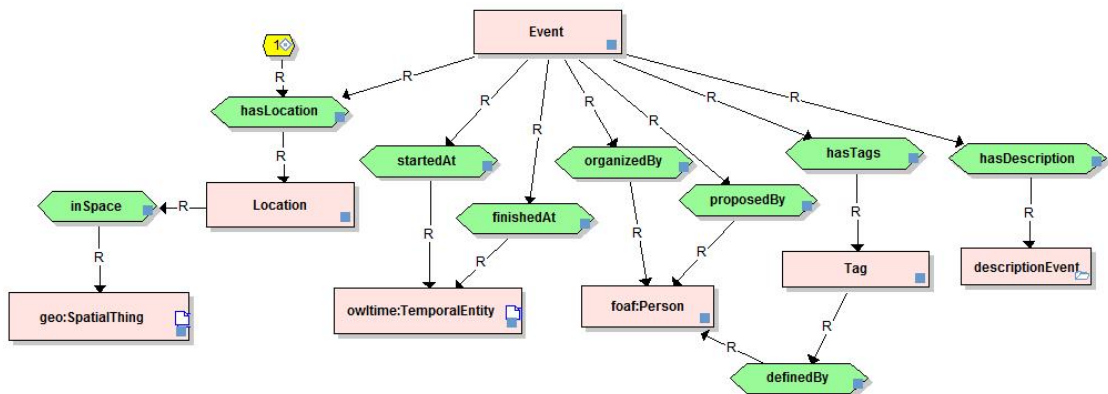


FIGURE 6.3 – Extrait de l'ontologie EventCCAlps.

L'ontologie CompanyCCAIPs

La Figure 6.4 présente un extrait de l'ontologie CompanyCCAIPs qui décrit les caractéristiques des entreprises participantes au projet CCAIPs (les partenaires) et aussi des participants aux événements organisés par le projet.

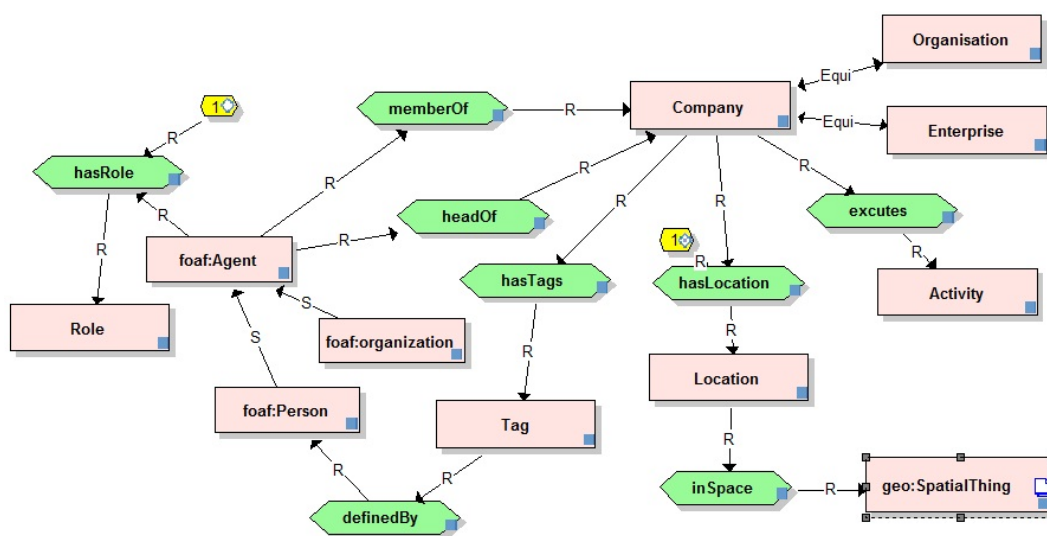


FIGURE 6.4 – Extrait de l'ontologie CompanyCCAIPs.

6.2.2 Changements ontologiques dans les ontologies CCAIPs

Durant leur cycle de vie, les ontologies CCAIPs ont subi plusieurs changements à la fois au niveau structurel et au niveau assertional. Ces changements sont de différents types : élémentaires, composés et complexes. Nous décrivons ci-dessous quelques exemples. La liste des changements ontologiques générés suite à l'application de ces cas est présentée dans le Tableau 6.1.

Exemple 1. Avec la diversité des événements organisés par le projet, une distinction entre leur type ("Conference", "Meeting", "Bootcamps") est devenue indispensable (Figure 6.5). Ce nouveau besoin engendre des changements au niveau de l'ontologie EventCCAIPs et nécessite l'ajout de nouvelles classes ("Conference", "Meeting", "Bootcamps"). Rappelons que pour éviter l'inconsistance de "concept isolé", l'ajout d'une nouvelle classe est accompagnée par une liste des changements dérivés (voir Chapitre 4, Section 4.4.1). Dans cet exemple, le changement

dérivé consiste à ajouter des axiomes de subsomption reliant les nouvelles classes à la classe "Event".

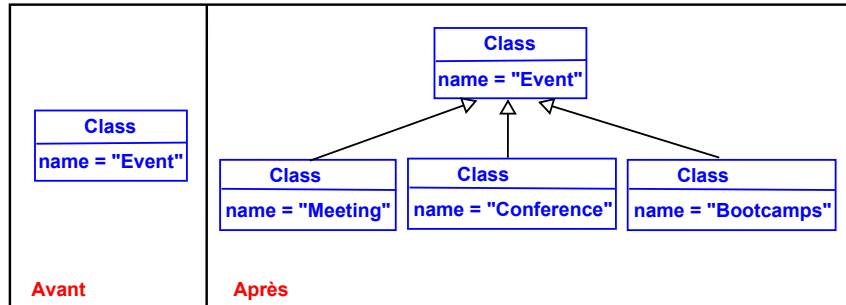


FIGURE 6.5 – Exemple 1 : ajout des classes et des axiomes de subsomption.

Exemple 2. Le projet organise souvent des événements, ce qui nécessite de peupler l'ontologie EventCCAlps par des nouvelles instances et leur assertions. Nous présentons à titre d'exemple l'ajout de l'événement "The Book of the Future" de type "Bootcamps" qui s'est organisé l'année 2013 à "Torino" (Figure 6.6).

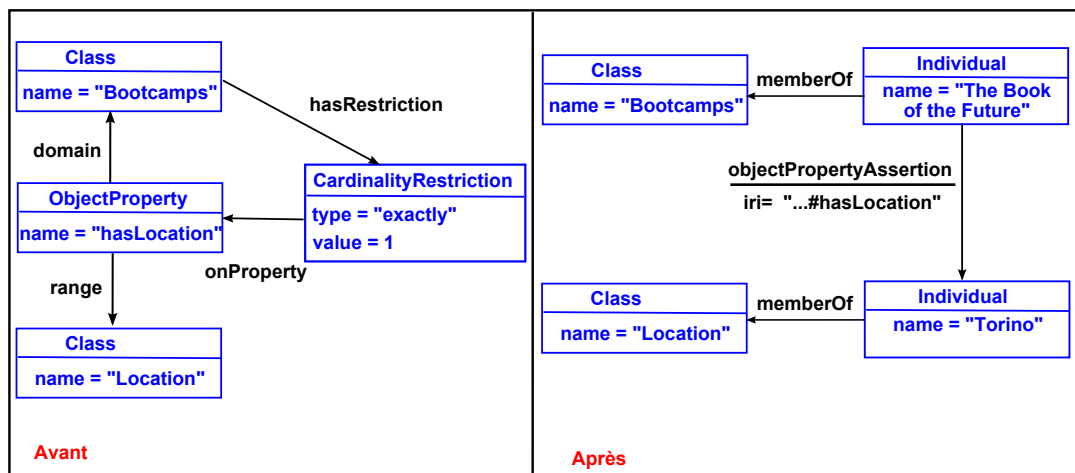


FIGURE 6.6 – Exemple 2 : ajout d'individus et d'assertions.

Exemple 3. Avec l'évolution du projet, les partenaires ont demandé une distinction entre les types d'entreprises (créatives et non créatives). Il faut donc remplacer la classe "Company" par les deux classes "CCI" et "NotCCI" sachant que CCI (Creative and Cultural Industries) sont les entreprises dont les activités concernent la créativité individuelle. Ce besoin d'évolution correspond au changement SplitClass et nécessite un ensemble de changements dérivés (voir Chapitre 4, Section 4.5.3) permettant de créer les nouvelles classes et les relier aux voisins de la classe "Company".

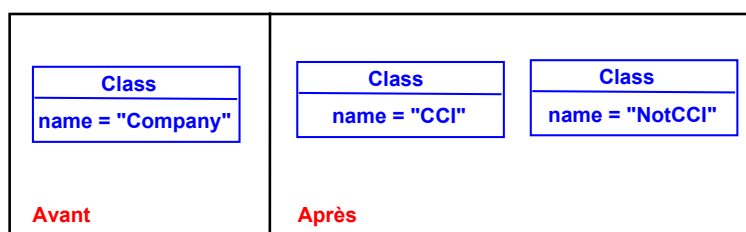


FIGURE 6.7 – Exemple 3 : diviser une classe.

Changements ontologiques	Application aux ontologies CCAIps
AddClass (CNew)	AddClass ("Meeting") AddClass ("Conference") AddClass ("Bootcamps")
AddSubClass (C1, C2)	AddSubClass ("Meeting", "Event") AddSubClass ("Conference", "Event") AddSubClass ("Bootcamps", "Event")
AddIndividual (C, INew)	AddIndividual ("Bootcamps", "The Book of the Future") AddIndividual ("Location", "Torino")
AddObjectPropertyAssertion (I1, I2, OB)	AddObjectPropertyAssertion ("The Book of the Future", "Torino", "hasLocation")
SplitClass(C, CNew1, CNew2)	SplitClass ("Company", "CCI", "notCCI")

TABLE 6.1: Exemple des changements dans les ontologies CCAIps.

6.3 Implémentation

Rappelons que notre formalisme de représentation et de gestion des changements ontologiques se base sur les grammaires de graphes et l'approche algébrique de Simple Pushout (SPO). Afin d'implémenter le concept de pushout et les différentes approches proposées (EvOGG et GROM), nous avons choisi de nous baser sur l'outil AGG (Attributed Graph Grammars). En effet, AGG supporte les approches algébriques, offre une représentation visuelle des graphes et des règles de réécritures et fournit aussi une API permettant son intégration dans des applications Java (voir le Chapitre 3, Section 3.4.4).

6.3.1 Les outils OWLToGGX et GGXToOWL

Afin de représenter des ontologies OWL selon le formalisme de grammaires de graphes typés, nous avons développé les outils OWLToGGX et GGXToOWL pour transformer des ontologies OWL en GGX et vice-versa. À noter que GGX est le format de représentation des graphes dans l'outil AGG.

L'outil OWLToGGX

OWLToGGX³ est un outil java basé sur l'API AGG⁴ et l'API Jena⁵ (une API open source permettant de lire et de manipuler des ontologies représentées en RDF, RDFS et OWL). OWLToGGX permet de transformer une ontologie OWL en un graphe hôte en s'appuyant sur le graphe type (*GT*) défini au chapitre 4 (Section 4.2.1). La transformation est décrite par l'algorithme 2.

3. Les codes sources sont disponibles en téléchargement, sous licence open source, via <http://mariem-mahfoudh.info/ksem2013/>

4. user.cs.tu-berlin.de/~gragra/agg

5. jena.sourceforge.net

Algorithme 2 : Transformation d'une ontologie OWL en un graphe AGG.**Entrées** : Ontologie (O), OWLMetamodel (G_T)**Sorties** : Graphe hôte (G)

Analyser O et extraire ses composants (ClassList, ObjPropertyList, DataPropertyList ...)

```

/* Ajouter les classes et leurs individus */
pour C in ClassList faire
  NC ← createNode("Class", C) ; NC.addAttribute(name) ;
  NC.addAttribute(iri) ; G.addNode(NC) ;
  IndList ← C.listIndividual();
  pour I in IndList faire
    NI ← G.createNode("Individual", I);
    G.addArc (NI, NC, "memberOf");
/* Ajouter les axiomes des classes */
pour C in ClassList faire
  subClassList ← C.listSubClass();
  pour subC in subClassList faire
    G.addArc (C, G.getNodeTypeClass(subC), "subClassOf");
  disjointClassList ← C.listDisjointClass() ; ...
/* Ajouter les objectProperties */
pour OB in ObjPropertyList faire
  NOB ← G.createNode("ObjectProperty", OB);
  domainList ← OB.domainList();
  pour dom in domainList faire
    G.addArc (NOB, G.getNodeTypeClass(dom), "domain");
  rangeList ← OB.rangeList();
  pour range in rangeList faire
    G.addArc (NOB, G.getNodeTypeClass(range), "range");
/* Ajouter les axiomes des objectProperties... */

```

La Figure 6.8 illustre un extrait de résultat de la transformation de l'ontologie EventCCAIPs par l'outil OWLToGGX. À noter que, pour des raisons de lisibilité, les IRIs ont été supprimés de la figure.

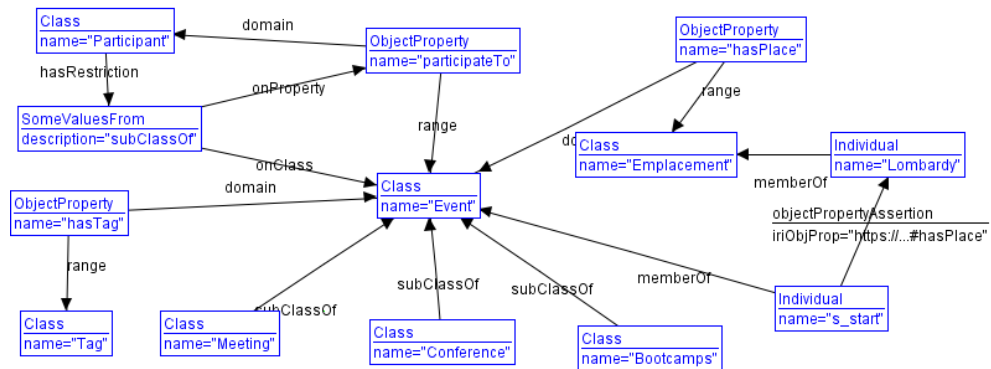


FIGURE 6.8 – Extrait de l'ontologie EventCCAtps.

L'outil GGXtoOWL

L'outil GGXtoOWL⁶ permet de transformer un graphe en format GGX en une ontologie OWL (voir algorithme 3). En effet, nous avons développé cet outil pour fournir la possibilité d'y tourner au langage OWL et être ainsi compatible avec les standards.

Algorithme 3 : Transformation du graphe AGG en une ontologie OWL.

Entrées : Graphe G

Sorties : Ontologie O

Parcourir G et extraire NodeList et EdgeList ;

pour *N* in NodeList **faire**

NType ← *N*.getType() ; *Niri* ← *N*.getAttribute(iri) ;

si *NType* = "Class" **alors**

 O.createClass(iri) ;

sinon si *NType* = "ObjectProperty" **alors**

 O.createObjectProperty(iri) ;

 ...

pour *E* in EdgeList **faire**

EType ← *E*.getType() ; *ESource* ← *E*.getSource() ;

ETarget ← *E*.getTarget() ;

suiant *EType* **faire**

 O.createAxiom (O.getElement(*ESource*), O.getElement(*ETarget*)) ;

6. <http://mariem-mahfoudh.info/ksem2013>

6.3.2 Approche EvOGG

L'implémentation de l'approche d'évolution d'ontologies EvOGG (Evolving Ontologies with Graph Grammars), décrite dans le chapitre 4, est basée sur l'outil AGG. Ainsi, avant de faire évoluer une ontologie, il faut qu'elle soit représentée par le formalisme de grammaires de graphes typés (Figure 6.9). Deux solutions sont envisagées pour l'utilisateur : 1) créer une ontologie (un graphe hôte) par l'interface graphique de l'outil AGG en chargeant le graphe type proposé⁷, ou bien, 2) transformer son ontologie OWL par l'outil OWLToGGX.



FIGURE 6.9 – Toolchain pour l'évolution d'ontologies.

La Figure 6.10 présente l'interface graphique de l'outil AGG et présente comment implémenter des changements ontologiques selon le formalisme proposé. Un exemple d'une grammaire de graphe est aussi illustré. Il est nommé "GraphTransformationSystem" et se compose de :

1. un graphe type qui correspond au méta-modèle d'ontologies ;
2. un graphe hôte présentant l'ontologie à faire évoluer ;
3. des règles de réécritures correspondant aux différents changements ontologiques.

Ainsi, pour faire évoluer son ontologie, l'utilisateur est invité à charger une ontologie et choisir la règle de réécriture qu'il veut appliquer.

Après chaque application de changement, un fichier de journalisation est généré permettant de sauvegarder l'ontologie et la règle de réécriture appliquée. Ceci permet notamment de garder une trace des différentes versions de ontologie et des différents changements qu'elle a subit. L'Annexe 2 présente un extrait d'un fichier de journalisation.

7. Le graphe type est disponible en téléchargement via <http://mariam-mahfoudh.info/src/kbs2014/metamodelOwl.ggx>

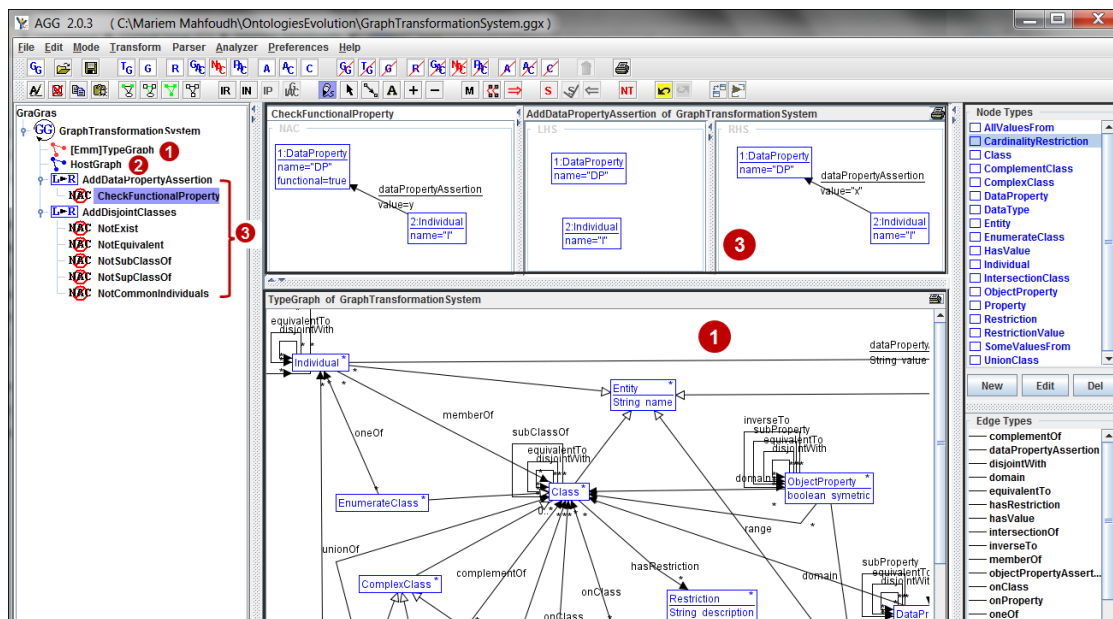


FIGURE 6.10 – Interface graphique de l'outil AGG.

6.3.3 Approche GROM

L'approche de fusion d'ontologies GROM (Graph Rewriting for Ontology Merging), détaillée dans la chapitre 5, est implémentée par deux outils⁸ (Figure 6.11) :

- l'outil OntologiesMapping permet d'identifier la similarité syntaxique et sémantique entre deux ontologies. Il est développé en Java et basé sur l'ontologie linguistique WordNet et la distance Levenshtein. Il prend en entrée deux ontologies et génère en sortie un fichier de mapping XML. À noter que cet outil, dans son état actuel, ne gère pas les relations structurelles ; celles-ci étant identifiées et ajoutées manuellement au fichier de mapping par un expert.
- l'outil GROM permet, d'une manière automatique, de fusionner deux ontologies selon le formalisme de grammaire de graphes typés. Il est lui aussi implémenté en Java et basé sur l'API AGG ce qui permet de réaliser les SPOs requis pour la fusion. L'outil prend en entrée deux ontologies au format GGX et leur fichier de mapping XML pour générer en sortie l'ontologie globale en format GGX.

8. les outils sont téléchargeable en Open Source via le lien <http://mariem-mahfoudh.info/medi2014/>

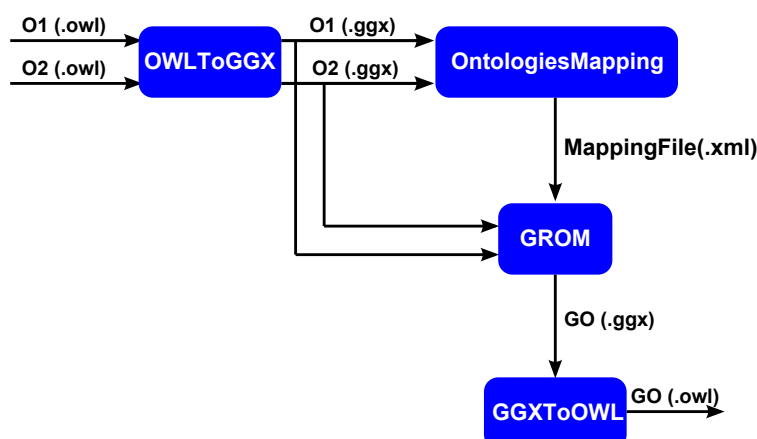


FIGURE 6.11 – Processus de fusion d'ontologies GROM.

6.4 Résultats et évaluation

6.4.1 Transformation d'ontologies OWL en graphes hôtes

Pour vérifier les résultats obtenus suite aux transformations réalisées par les outils OWLToGGX et GGXToOWL, nous nous sommes basés sur les statistiques fournies par les éditeurs d'ontologies Protégé⁹ [Knublauch et al., 2004] et swoop¹⁰ [Kalyanpur et al., 2006]. Le test consiste tout simplement à transformer une ontologie OWL (O) par l'outil OWLToGGX. Ensuite, nous transformons le graphe hôte généré par l'outil GGXToOWL pour obtenir de nouveau une ontologie OWL (O_{transf}). La comparaison est réalisée ainsi entre les deux ontologies O et O_{transf} pour vérifier si la transformation a été réalisée avec ou sans perte d'information. Le test a été réalisé non seulement sur les ontologies CCAIps mais aussi sur plusieurs autres ontologies comme celles publiées dans la bibliothèque de Protégé¹¹. La Figure 6.12 présente les métriques affichées par l'outil swoop confirmant la préservation des concepts et des axiomes de l'ontologie "Travel" après sa transformation par les outils OWLToGGX et GGXToOWL.

6.4.2 Approche d'évolution

L'approche d'évolution proposée a été appliquée sur les ontologies CCAIps pour suivre leurs différents changements. Rappelons que notre approche est une approche de résolution a priori des inconsistances. La qualité de l'ontologie évoluée est préservée grâce à l'utilisation

9. protege.stanford.edu

10. github.com/ronwalf/swoop

11. protegewiki.stanford.edu/wiki/Protege_Ontology_Library

<p>OWL Ontology: travel.owl</p> <p>Annotations: rdfs:comment (Datatype http://www.w3.org/2001/XMLSchema#string) : An example ontology for tutorial purposes. owl:versionInfo (Datatype http://www.w3.org/2001/XMLSchema#string) : 1.0 by Holger Knublauch (holger@smi.stanford.edu)</p> <p>Total Number of Classes: 34 (Defined: 34, Imported: 0) Total Number of Datatype Properties: 4 (Defined: 4, Imported: 0) Total Number of Object Properties: 6 (Defined: 6, Imported: 0) Total Number of Annotation Properties: 2 (Defined: 2, Imported: 0) Total Number of Individuals: 14 (Defined: 14, Imported: 0)</p>		
Advanced Ontology Statistics:		
General Statistics	Property Tree Statistics	Satisfiable Class Tree Statistics
DL Expressivity: SION(D) No. of GCIs: 0 No. of Sub-classes: 24 No. of Disjoint Axioms: 10 No. of Functional Properties: 4 No. of Inverse Functional Properties: 0 No. of Transitive Properties: 1 No. of Symmetric Properties: 0 No. of Inverse Properties: 2	Properties with Multiple Inheritance: 0 Max. Depth of Property Tree: 2 Min. Depth of Property Tree: 2 Avg. Depth of Property Tree: ?	Classes with Multiple Inheritance: 1 Max. Depth of Class Tree: 4 Min. Depth of Class Tree: 1 Avg. Depth of Class Tree: 2.25 Max. Branching Factor of Class Tree: 11 Min. Branching Factor of Class Tree: 1 Avg. Branching Factor of Class Tree: 2.9
<p>OWL Ontology: travelAfterTransfo.owl</p> <p>Annotations:</p> <p>Total Number of Classes: 34 (Defined: 34, Imported: 0) Total Number of Datatype Properties: 4 (Defined: 4, Imported: 0) Total Number of Object Properties: 6 (Defined: 6, Imported: 0) Total Number of Annotation Properties: 0 (Defined: 0, Imported: 0) Total Number of Individuals: 14 (Defined: 14, Imported: 0)</p>		
Advanced Ontology Statistics:		
General Statistics	Property Tree Statistics	Satisfiable Class Tree Statistics
DL Expressivity: SION(D) No. of GCIs: 18 No. of Sub-classes: 24 No. of Disjoint Axioms: 10 No. of Functional Properties: 4 No. of Inverse Functional Properties: 0 No. of Transitive Properties: 1 No. of Symmetric Properties: 0 No. of Inverse Properties: 2	Properties with Multiple Inheritance: 0 Max. Depth of Property Tree: 2 Min. Depth of Property Tree: 2 Avg. Depth of Property Tree: ?	Classes with Multiple Inheritance: 1 Max. Depth of Class Tree: 4 Min. Depth of Class Tree: 1 Avg. Depth of Class Tree: 2.25 Max. Branching Factor of Class Tree: 11 Min. Branching Factor of Class Tree: 1 Avg. Branching Factor of Class Tree: 2.9

FIGURE 6.12 – Comparaison des ontologies Travel (avant et après transformation) par l’outil SWOOP

des conditions d’applications négatives (NAC) qui interdisent l’application de tout changement susceptible de causer des inconsistances. Comme exemple, nous présentons le changement *AddDisjointClasses(Meeting, Event)* appliqué sur l’ontologie EventCCalps (voir Figure 6.13).

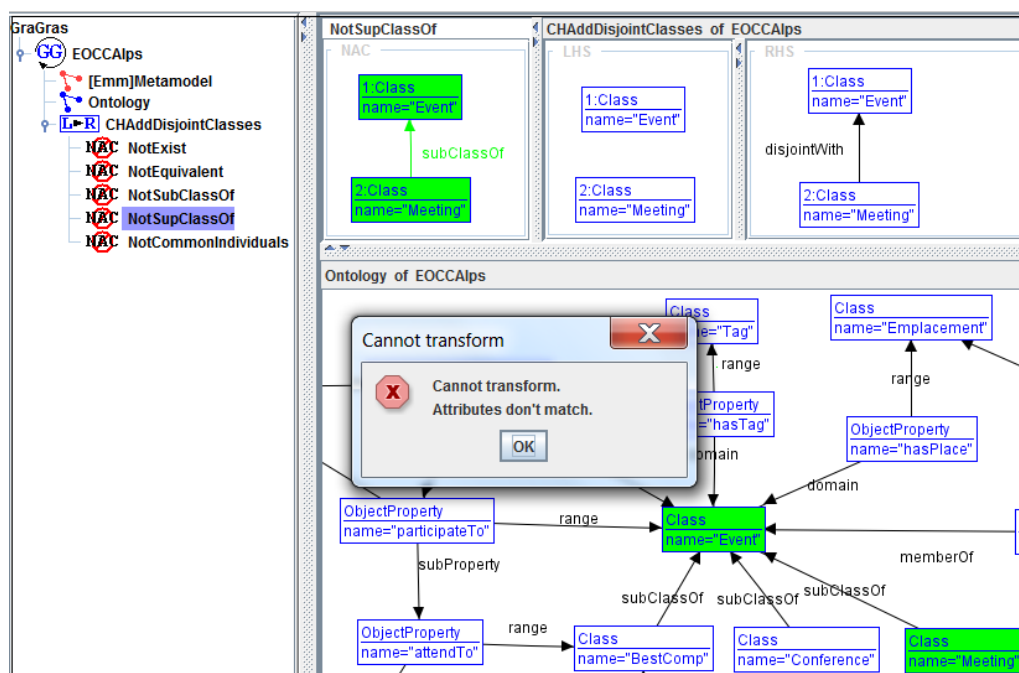


FIGURE 6.13 – Implémentation du changement *AddDisjointClasses(Meeting, Event)*.

Le changement *AddDisjointClasses(Meeting, Event)* permet d'ajouter un axiome de disjonction entre les deux classes "Meeting" and "Event" et sa règle de réécriture contient cinq NAC :

1. NAC 1 permettant d'éviter la redondance ;
2. NAC 2 permettant d'interdire l'application du changement si les classes "Meeting" et "Event" sont des classes équivalentes (deux classes ne peuvent pas être à la fois équivalentes et disjointes) ;
3. NAC 3 permettant d'interdire la transformation si la classe "Event" est une sous-classe de la classe "Meeting" (deux classes partageant une relation de subsomption ne peuvent être disjointes) ;
4. NAC 4 permettant d'interdire la transformation si la classe "Meeting" est une sous-classe de la classe "Event" ;
5. NAC 5 permettant d'interdire la transformation si les deux classes possèdent des individus communs.

Comme les classes "Meeting" et "Event" partagent une relation de subsomption (la classe "Meeting" est sous classe de la classe "Event"), la transformation ne peut pas être réalisée (violation du NAC 4) et une alerte est affichée (comme le montre la Figure 6.13) pour notifier l'utilisateur que la transformation ne peut pas être achevée.

Complexité. Avec les différents tests à vérifier pour l'application des changements ontologiques, une question légitime peut se poser sur la complexité du formalisme défini. En effet, le temps d'application d'une règle de réécriture dépend à la fois de la taille de *LHS* et des *NAC*. L'étape la plus coûteuse en temps et en ressource est l'identification des graphes patterns depuis le graphe hôte. Il s'agit d'un problème NP-complet (une recherche d'un sous-graphe composé de k éléments dans un graphe composé de n élément est de complexité $O(n^k)$). Ainsi, le coût de calcul reste tout à fait acceptable si les tailles de *LHS* et *NAC* sont limitées [Karsai et al., 2003]. Cette condition est satisfaite dans la plupart des règles de réécriture définissant les changements ontologiques. Nous présentons dans le Tableau 6.2 la taille des *LHS* de certains changements ontologiques. Nous pouvons dire qu'elle est très réduite pour la plupart des changements ontologiques simples (0 à 3 nœuds). Pour les changements ontologiques composées et complexes, il faut aussi ajouter la taille des LHS de leurs changements dérivés (*CHD*).

Changement ontologique	Taille de LHS
<i>AddIndividual</i>	0 nœud
<i>RenameClass</i> , <i>RenameDataProperty</i> , <i>RenameIndividual</i>	1 nœud
<i>AddEquivalentClasses</i> , <i>AddSubClasses</i> , <i>AddDisjointClasses</i>	2 nœuds
<i>AddObjectProperty</i> , <i>AddDataProperty</i>	3 nœuds
<i>PullUpClass</i>	Au moins 3 nœuds pour le changement principal. Au moins 4 nœuds pour le premier changement dérivé <i>CHD</i> . Au moins 6 nœuds pour le deuxième changement dérivé <i>CHD</i> .

TABLE 6.2: Taille des LHS de certains changements ontologiques.

6.4.3 Approche de fusion

Valider une approche de fusion d'ontologies est une tâche difficile [Raunich and Rahm, 2012] qui dépend de plusieurs facteurs :

- la qualité des ontologies à fusionner (consistantes ou bien inconsistantes) ;
- la qualité de mapping utilisé (couvre-t-il les correspondances syntaxiques et/ou sémantiques et/ou structurelles,...) ;

- la stratégie adoptée pour la fusion (symétrique ou bien asymétrique), etc.

Dans notre travail, nous considérons que les ontologies initiales sont consistantes. Nous nous positionnons dans le cadre des approches asymétriques et nous nous basons sur les métriques suivantes pour évaluer la qualité de résultat de la fusion :

1. *la redondance* permet d'identifier les concepts qui se répètent dans l'ontologie globale. Une approche de fusion doit éviter ou bien minimiser la redondance dans le résultat globale [Raunich and Rahm, 2012]. Dans notre travail, la non redondance de données est assurée grâce à l'utilisation des *NAC* ;
2. *la consistence/cohérence*, vérifie la satisfaction de l'ontologie globale par rapport à un modèle donné. Dans notre approche de fusion, les inconsistances sont gérées par les règles de réécriture et leurs conditions d'application, notamment les *NAC*.
3. *la couverture*, appelée en anglais *coverage*, est le degré de préservation des concepts des ontologies source et destination dans le résultat de la fusion. Elle varie entre 0 et 1 et est calculée comme suit : $Couverture = card(N(OG))/card(N(O1)+N(O2))$ [Raunich and Rahm, 2012].

Le Tableau 6.3 présente un ensemble d'ontologies¹² fusionnées avec l'outil GROM. Il précise leur nombre de concepts, leurs correspondances (nombre des *NC*, *NE*, *NS* et *NIsa*), le nombre de concepts du résultat de leur fusion, le nombre de concepts répétés (redondance) , le nombre des inconsistances détectées dans le résultat de fusion et aussi la valeur de couverture. Si nous prenons le premier exemple (ontologies CCAIps), nous voyons que la couverture est égale 0.9 et l'ensemble de correspondances entre les deux ontologies est égale à 4 : une paire de nœuds communs, une paire de nœuds équivalents, une paire de nœuds partageant une relation sémantique et une paire de nœuds partageant une relation de subsomption. Ceci reflète que les éléments redondants (*NC*) ont été supprimés dans le résultat de fusion et que les nœuds équivalents (*NE*) ont été remplacés. Dans cet exemple, les relations d'équivalences (*NS*) et de subsomptions (*NIsa*) ont été ajoutés à l'ontologie globale puisqu'elles n'altèrent pas sa consistance.

12. Toutes les ontologies (en OWL et en GGX) et l'implémentation Java sont disponibles en téléchargement en open source via : <http://mariam-mahfoudh.info/ontologiesBenchmark/>

Ontologies	#Concepts	#Correspondances	#Fusion	Redondance	Inconsistances	Couverture	Utilisées dans
CCAAlps (Event, Company)	12 .. 8	4 #NC=1, #NE=1, #NS=1, #Nisa=1	18	0	0	0.9	<i>Matfouadh et al.</i> [2014c]
Cars	7 .. 6	6 #NC=3, #Nisa =3	13	0	0	1.00	<i>Raunich and Rahm</i> [2012]
Lebensmittel (Google, web)	53 .. 59	15 #NS=15	112	0	0	1.00	<i>Peukert et al.</i> [2010]
Freizeit (dmoz, Google)	71.. 67	67 #NE=67	71	0	0	0.51	<i>Peukert et al.</i> [2010]
Conference (ekaw, iasted) ¹³	107 .. 182	10 #NC=5, #NE=3, #NS=2	280	0	1	0.97	-
Conference (cmt, confOf) ¹⁴	30 .. 39	9	280	0	0	0.84	-

TABLE 6.3: Fusion de certaines ontologies avec l'outil GROM.

13. <http://oaei.ontologymatching.org/2013/conference/>14. <http://oaei.ontologymatching.org/2013/conference/>

6.5 Bilan

Dans ce chapitre, nous avons présenté le cadre applicatif de notre travail. Nous avons introduit les outils développés pour valider les différentes approches proposées. Nous avons également décrit des exemples d'applications et les résultats obtenus sur un ensemble d'ontologies (essentiellement sur celles développées dans le cadre du projet CCAIps). Ce chapitre montre ainsi que l'utilisation des grammaires de graphes assure à la fois une formalisation rigoureuse pour la spécification des changements et offre une variété d'outils facilitant l'implémentation des formalisations définies.

Contributions et valorisation

Les ontologies CCAIps et les outils développés ont été présentés à la communauté à travers 3 communications (2 revues internationales et une conférence internationale). La première communication [Mahfoudh et al., 2014c] dans une conférence internationale (*International Conference on Model & Data Engineering*). La deuxième [Thiry et al., 2014] a été aussi publiée dans un journal international (*International Journal of Web Applications*). La troisième [Mahfoudh et al., 2015a] a été publié dans un journal international (*Knowledge-Based Systems*).

Chapitre 7

Conclusion

Sommaire

7.1 Contributions	142
7.2 Perspectives	143
7.2.1 Perspectives à court terme	143
7.2.2 Perspectives à moyen terme	144
7.2.3 Perspectives à long terme	145

*"Un jour nous serons ce que nous voulons.
Le voyage n'a pas commencé, le chemin n'est pas achevé,"*
Mahmoud Darwich

Dans cette thèse, nous avons abordé l'utilisation des grammaires de graphes typés pour la formalisation et l'implémentation des changements d'ontologies. Dans une première partie, nous avons présenté les processus d'évolution et de fusion d'ontologies en mettant la lumière sur leurs enjeux. Ensuite, nous avons introduit le formalisme de grammaires de graphes typés et les différentes approches et outils permettant la transformation des graphes. L'objectif étant de montrer les points communs entre ces deux formalismes afin de les coupler et tirer parti des avantages des grammaires de graphes pour répondre aux problématiques des changements d'ontologies.

La deuxième partie de la thèse s'est consacrée à la présentation de nos propositions. En premier lieu, nous avons introduit une approche d'évolution d'ontologies, avec une résolution a priori des inconsistances, basée sur les concepts de grammaires de graphes. Une formalisation et une classification des différents types de changements ontologiques ont été également présentées. En deuxième lieu, nous avons décrit une approche asymétrique de fusion d'ontologies qui se base elle aussi sur les grammaires de graphes et les approches algébriques de

transformations de graphes. Enfin, une application de ces deux approches a été réalisée sur les ontologies CCAIps développées dans le cadre du projet Européen CCAIps qui finance les travaux de cette thèse.

7.1 Contributions

Les travaux de cette thèse présentent différentes contributions.

Nouveau formalisme de représentation d'ontologies. Nous avons proposé un nouveau formalisme de représentation d'ontologies qui est capable aussi de formaliser leurs changements [Mahfoudh et al., 2013b]. Ce formalisme, $TGGOnto = \{GT_O, G_O, R_O\}$ est défini par les grammaires de graphes typés et les approches algébriques de transformation de graphes. Les ontologies sont représentées par des graphes hôtes G_O dont les types de concepts sont spécifiés par un graphe type GT_O . Les changements d'ontologies sont formalisés par des règles de réécriture R_O .

Approche d'évolution d'ontologies. Les travaux de cette thèse ont donné naissance à une approche d'évolution d'ontologies EvOGG (Evolving Ontologies with Graph Grammars) qui est basée sur notre formalisme TGGOnto. Cette approche se démarque des approches de la littérature par le traitement a priori des inconsistances permettant ainsi de préserver la consistance de l'ontologie évoluée [Mahfoudh et al., 2015a]. Cette caractéristique est assurée grâce à l'utilisation des Conditions d'Applications Négatives (NAC) des grammaires de graphes. De plus, notre approche traite à la fois : (1) de l'enrichissement d'ontologies en étudiant le niveau structurel d'ontologies et (2) du peuplement d'ontologies en étudiant les changements qui affectent les individus et leurs assertions. L'approche EvOGG définit des changements ontologiques de différents types : élémentaires [Mahfoudh et al., 2013a], composés et complexes [Mahfoudh et al., 2015b] et assure leur implémentation par l'approche algébrique Simple PushOut (SPO) de transformation de graphes. Dans le chapitre 4, nous avons montré aussi le formalisme des grammaires de graphes défini permet de diminuer le nombre de règles définissant certains changements ontologiques considérés dans la littérature comme changement composés (ou complexes).

Approche de fusion d'ontologies. En se basant sur le formalisme TGGOnto et l'approche EvOGG, nous avons proposé une approche de fusion d'ontologies GROM (Graph Rewriting for Ontologies Merging) [Mahfoudh et al., 2014c]. Notre approche est asymétrique et est capable d'éviter les redondances de données et de diminuer les conflits dans l'ontologie globale

(résultant du processus de la fusion). L'approche GROM assure la fusion d'ontologies par l'approche algébrique SPO permettant ainsi de simplifier l'intégration des ontologies sources et destination [Mahfoudh et al., 2014b].

Développement logiciel. En complément des modèles théoriques précédents, nous avons aussi développé plusieurs outils Open Source pour mettre en œuvre les différentes approches proposées : (1) l'outil OWLToGGX permet de transformer une ontologie OWL sous format d'un graphe hôte compréhensible par le formalisme de grammaires de graphes; (2) l'outil GGXToOWL permet de transformer une ontologie représentée en graphe hôte au standard OWL; (3) l'outil GROM (Graph Rewriting for Ontology Merging) permet de fusionner d'une manière automatique deux ontologies; (4) l'implémentation des changements ontologiques de l'approche EvOGG avec l'outil AGG (Attributed Graph Grammar).

7.2 Perspectives

Les travaux présentés dans cette thèse ouvrent de nombreuses perspectives à court, à moyen et à long terme.

7.2.1 Perspectives à court terme

Plugin EvOGG pour l'éditeur Protégé. Comme déjà évoqué dans le chapitre 6, l'approche d'évolution EvOGG et les différents changements ontologiques sont implémentés en se basant sur l'outil AGG. Nous travaillons actuellement sur l'implémentation de ces changements (simples, composés et complexes) par l'API java d'AGG afin de développer un plugin pour l'éditeur Protégé. En effet, ceci offre à l'utilisateur une meilleure flexibilité et lui permet de travailler directement sur les standards sans avoir besoin des connaissances sur les outils des grammaires de graphes. Ainsi, les outils OWLToGGX et GGXToOWL seront intégrés au plugin EvOGG et les transformations d'ontologies OWL en AGG (et inversement) seront masquées à l'utilisateur.

Versionning d'ontologies. L'application des changements ontologiques pose des questions sur le sort des différentes versions d'ontologies obtenues après chaque transformation. A l'heure actuelle, nous sauvegardons toutes les versions des ontologies avec les règles de réécritures appliquées. Ceci permet notamment de garder une trace de tous les changements réalisés et d'identifier la différence entre les ontologies. Cependant, aucune optimisation sur leur stockage n'est réalisée. Une étude en profondeur sur la problématique du versionnage d'ontologies devra être réalisée pour savoir quelles sont les versions qu'il faut garder et pourquoi.

Décomposition d'ontologies. La décomposition d'ontologies est un sujet de recherche aussi intéressant que celui de la fusion d'ontologies. Elle consiste à décomposer des ontologies en modules et aussi de trouver des ontologies sources à partir d'une ontologie globale. Ceci intéresse bien entendu les chercheurs travaillant sur l'extraction des modules, notamment dans les ontologies modulaires. Dans notre travail, nous avons formalisé la fusion par l'opérateur de pushout de la théorie des catégories. Sachant que l'opérateur pullback est le dual de pushout (voir Chapitre 3, Section 3.3.1), nous envisageons d'étudier la décomposition d'ontologies par l'opérateur pullback. Quant à l'extraction des modules, elle pourra être réalisée facilement par des règles de réécritures spécifiant le module recherché.

Ontologies interdépendantes. Dans l'approche d'évolution d'ontologies, nous avons traité la propagation d'un changement ontologique vers les concepts et les individus (artéfacts dépendants) de l'ontologie évoluée. Sachant que nous avons étudié la problématique de la fusion d'ontologies, il serait intéressant de traiter aussi la propagation d'un changement vers les ontologies reliées à l'ontologie évoluée. En d'autres termes, si une ontologie subit un changement, comment mettre à jour l'ontologie globale et les autres ontologies reliées afin de préserver la consistance de toutes les ontologies considérées.

7.2.2 Perspectives à moyen terme

Les perspectives à moyen terme touchent les domaines du traitement automatique des langues et la fouille de données en relation avec nos travaux.

Identification automatiques des changements ontologiques. L'identification des changements ontologiques est la première phase dans le processus d'évolution d'ontologies. Elle consiste à déterminer les changements nécessaires pour exprimer les besoins d'évolution. Actuellement, dans notre approche EvOGG, c'est l'utilisateur qui choisit les changements à apporter à l'ontologie. Nous envisageons d'automatiser cette étape en faisant appel aux techniques de fouille de données et de traitement automatique des langues. En effet, les besoins d'évolution peuvent être capturés en analysant des versions d'ontologies, des profils d'utilisateurs, des corpus de textes décrivant les ontologies évoluées, etc.

Automatiser la recherche de similarité entre ontologies. Rappelons que dans notre approche de fusion d'ontologies GROM, nous nous sommes intéressés essentiellement à l'étape de fusion d'ontologies. L'automatisation de l'étape de recherche de correspondances entre les ontologies n'était pas parmi les objectifs principaux de notre travail. En effet, l'outil OntologiesMapping développé ne couvre, dans son état actuel, que certains types de similarités

syntaxiques et sémantiques. Nous envisageons d'étudier d'une manière plus approfondie les différentes mesures de similarité, notamment celles qui s'intéressent à la structure des ontologies, afin d'automatiser cette étape.

7.2.3 Perspectives à long terme

Langage d'interrogation pour les ontologies OWL. Les travaux menés dans cette thèse peuvent présenter le point de départ pour la proposition d'un nouveau langage d'interrogation d'ontologie OWL basé sur les grammaires de graphes. En effet, dans la littérature, un travail s'est intéressé à l'utilisation des grammaires de graphes pour l'interrogation des graphes RDF [*Braatz and Brandt, 2008*]. Cependant, à ce jour l'idée n'a pas été implémentée. Nous pensons que grâce au formalisme TGGOnto, l'outil OWLToGGX et les différentes règles de réécriture que nous avons formalisées et développées, une définition d'un langage d'interrogation pour les ontologies OWL pourra voir le jour.

Annexes

Annexe A

Formalisation des changements ontologiques avec les grammaires de graphes

Nous présentons dans cette annexe la formalisation des changements ontologiques étudiés durant cette thèse. Nous précisons pour chaque changement ses NAC, son LHS, son RHS et ses CHD.

Changement ontologique	NAC	LHS	RHS	CHD
Changements élémentaires : changements de renommage				
<i>RenameClass</i> (C_i, C_{New})	$\{C_{New}\}$	$\{C_i\}$	$\{C_{New}\}$	—
<i>RenameIndividual</i> (I_i, I_{New})	$\{I_{New}\}$	$\{I_i\}$	$\{I_{New}\}$	—
<i>RenameObjectProperty</i> - (OP_i, OP_{New})	$\{OP_{New}\}$	$\{OP_i\}$	$\{OP_{New}\}$	—
<i>RenameDataProperty</i> - (DP_i, DP_{New})	$\{DP_{New}\}$	$\{DP_i\}$	$\{DP_{New}\}$	—
Changements élémentaires : changements d'ajout				
<i>AddIndividual</i> (I_{New}, C_i)	$\{I_{New}\}$	$\{C_i\}$	$\{I_{New} \in C_i\}$	—
<i>AddObjectProperty</i> (OP_{New}, C_1, C_2)	$\{OP_{New}\}$	$\{C_1, C_2\}$	$\{OP_{New} \sqsubseteq C_1$ $\wedge \top \sqsubseteq \forall OB_{New}.C_2\}$	—
<i>AddDataProperty</i> (DP_{New}, C_i, D_i)	$\{DP_{New}\}$	$\{C_i, D_i\}$	$\{DP_{New} \sqsubseteq C_i$ $\wedge \top \sqsubseteq \forall DP_{New}.D_i\}$	—
<i>AddEquivalentClasses</i> (C_1, C_2)	1. $C_1 \sqsubseteq \neg C_2$; 2. $C_1 \equiv C_2$.	$\{C_1, C_2\}$	$\{C_1 \equiv C_2\}$	—
<i>AddDisjointClasses</i> (C_1, C_2)	1. $C_1 \sqsubseteq \neg C_2$; 2. $C_1 \equiv C_2$; 3. $C_1 \sqsubseteq C_2$; 4. $C_2 \sqsubseteq C_1$; 5. $\exists I_i \in I(O) \cdot I_i \in C_1$ $\wedge I_i \in C_2$.	$\{C_1, C_2\}$	$\{C_1 \sqsubseteq \neg C_2\}$	—

<i>AddSubClass</i> (C_1, C_2)	<ol style="list-style-type: none"> 1. $C_1 \sqsubseteq C_2$; 2. $C_2 \sqsubseteq C_1$; 3. $C_1 \sqsubseteq \neg C_2$; 4. $\exists C_i \in C(O) \cdot (C_1 \sqsubseteq C_i) \wedge (C_i \sqsubseteq C_2)$; 5. $\exists (C_i, C_j) \in C(O) \cdot (C_i \sqsubseteq C_1) \wedge (C_j \sqsubseteq C_2) \wedge C_i \sqsubseteq \neg C_j$. 	$\{C_1, C_2\}$	$\{C_1 \sqsubseteq C_2\}$	—
<i>AddEquivalentObjectProperties</i> (OP_1, OP_2)	<ol style="list-style-type: none"> 1. $OP_1 \sqsubseteq \neg OP_2$; 2. $OP_1 \equiv OP_2$. 	$\{OP_1, OP_2\}$	$\{OP_1 \equiv OP_2\}$	—
<i>AddDisjointObjectProperties</i> (OP_1, OP_2)	<ol style="list-style-type: none"> 1. $OP_1 \sqsubseteq \neg OP_2$; 2. $OP_1 \equiv OP_2$; 3. $OP_1 \sqsubseteq OP_2$; 4. $OP_2 \sqsubseteq OP_1$. 	$\{OP_1, OP_2\}$	$\{OP_1 \sqsubseteq \neg OP_2\}$	—
<i>AddSubObjectProperties</i> (OP_1, OP_2)	<ol style="list-style-type: none"> 1. $OP_1 \sqsubseteq OP_2$; 2. $OP_2 \sqsubseteq OP_1$; 3. $OP_1 \sqsubseteq \neg OP_2$; 	$\{OP_1, OP_2\}$	$\{OP_1 \sqsubseteq OP_2\}$	—
<i>AddEquivalentDataProperties</i> (DP_1, DP_2)	<ol style="list-style-type: none"> 1. $DP_1 \sqsubseteq \neg DP_2$; 2. $DP_1 \equiv DP_2$. 	$\{DP_1, DP_2\}$	$\{DP_1 \equiv DP_2\}$	—
<i>AddDisjointDataProperties</i> (DP_1, DP_2)	<ol style="list-style-type: none"> 1. $DP_1 \sqsubseteq \neg DP_2$; 2. $DP_1 \equiv DP_2$; 3. $DP_1 \sqsubseteq DP_2$; 4. $DP_2 \sqsubseteq DP_1$. 	$\{DP_1, DP_2\}$	$\{DP_1 \sqsubseteq \neg DP_2\}$	—
<i>AddSubDataProperties</i> (DP_1, DP_2)	<ol style="list-style-type: none"> 1. $DP_1 \sqsubseteq DP_2$; 2. $DP_2 \sqsubseteq DP_1$; 3. $DP_1 \sqsubseteq \neg DP_2$; 	$\{DP_1, DP_2\}$	$\{DP_1 \sqsubseteq DP_2\}$	—
<i>AddRangeDataTypeClass</i> (DP_i, D_i)	—	$\{DP_i, D_i\}$	$\{\top \sqsubseteq \forall DP_i. D_i\}$	—
<i>AddDomainDataProperty</i> (DP_i, C_i)	—	$\{DP_i, C_i\}$	$\{DP_i \sqsubseteq C_i\}$	—

<i>AddObjectPropertyAssertion</i> (I_1, I_2, OP_i)	<ol style="list-style-type: none"> 1. $(I_1, I_2) \in OP_i$; 2. $\exists I_i \in I(O) \cdot (I_i \neq I_2) \wedge ((I_1, I_i) \in OP_i) \wedge (\top \sqsubseteq \perp OP_i)$; 3. $\exists \leq n OP_i \cdot (\exists I_i \in I(O)) \wedge \{(I_1, I_i) \in OP_i\} = n$. 	$\{I_1, I_2, OP_i\}$	$\{(I_1, I_2) \in OP_i\}$	—
Changements élémentaires : changements de suppression				
<i>RemoveIndividual</i> (I_i)	—	$\{I_i\}$		—
<i>RemoveDataProperty</i> (DP_i)	—	$\{DP_i\}$		—
<i>RemoveDisjointClasses</i> (C_1, C_2)	—	$\{C_1 \sqsubseteq \neg C_2\}$	$\{C_1, C_2\}$	—
<i>RemoveEquivalentClasses</i> (C_1, C_2)	—	$\{C_1 \equiv C_2\}$	$\{C_1, C_2\}$	—
<i>RemoveSubClass</i> (C_1, C_2)	—	$\{C_1 \sqsubseteq C_2\}$	$\{C_1, C_2\}$	—
<i>RemoveDisjointObjectProperties</i> (OP_1, OP_2)	—	$\{OP_1 \sqsubseteq \neg OP_2\}$	$\{OP_1, OP_2\}$	—
<i>RemoveEquivalentObjectProperties</i> (OP_1, OP_2)	—	$\{OP_1 \equiv OP_2\}$	$\{OP_1, OP_2\}$	—
<i>RemoveSubObjectProperty</i> (OP_1, OP_2)	—	$\{OP_1 \sqsubseteq OP_2\}$	$\{OP_1, OP_2\}$	—
<i>RemoveDisjointDataProperties</i> (DP_1, DP_2)	—	$\{DP_1 \sqsubseteq \neg DP_2\}$	$\{DP_1, DP_2\}$	—
<i>RemoveEquivalentDataProperties</i> (DP_1, DP_2)	—	$\{DP_1 \equiv DP_2\}$	$\{DP_1, DP_2\}$	—
<i>RemoveSubDataProperty</i> (DP_1, DP_2)	—	$\{DP_1 \sqsubseteq DP_2\}$	$\{DP_1, DP_2\}$	—
<i>RemoveDomainDataProperty</i> (DP_i, C_i)	—	$\{DP_i \sqsubseteq C_i\}$	$\{DP_i, C_i\}$	—
<i>RemoveRangeDataType</i> (DP_i, D_i)	—	$\{\top \sqsubseteq \forall DP_i . D_i\}$	$\{DP_i, D_i\}$	—
<i>RemoveObjectPropertyAssertion</i> (I_1, I_2, OP_i)	—	$\{(I_1, I_2) \in OP_i\}$	$\{I_1, I_2, OP_i\}$	—
Changements composés : changements d'ajout				
<i>AddClass</i> (C_{New})	$\{C_{New}\}$	—	$\{C_{New}\}$	<ol style="list-style-type: none"> 1. <i>AddSubClass</i> 2. <i>AddDisjointClasses</i> 3. <i>AddEquivalentClasses</i> 4. <i>AddDomain</i> 5. <i>AddRange</i>
Changements composés : changements de suppression				

<i>RemoveClass</i> (C_i)	—	$\{C_i\}$	—	<ol style="list-style-type: none"> 1. <i>SetTypeIndividual</i> 2. <i>RemoveIndividual</i> 3. <i>RemoveRestrictions</i>
<i>RemoveObjectProperty</i> (OP_i)	—	$\{OP_i\}$	—	<ol style="list-style-type: none"> 1. <i>RemoveRestriction</i> 2. <i>RemoveAssertionObjectProperty</i>
<i>RemoveDataProperty</i> (DP_i)	—	$\{DP_i\}$	—	<ol style="list-style-type: none"> 1. <i>RemoveRestriction</i> 2. <i>RemoveAssertionDataProperty</i>
<i>RemoveCardinalityRestriction</i> (C_i, OP_i)	—	$\{C_i, OP_i\}$	—	<i>RemoveAssertionObjectProperty</i>
<i>RemoveSomeValuesRestriction</i> (C_i, OP_i)	—	$\{C_i, OP_i\}$	—	<i>RemoveAssertionObjectProperty</i>
<i>RemoveAllValuesFromRestriction</i> (C_i, OP_i)	—	$\{C_i, OP_i\}$	—	<i>RemoveAssertionObjectProperty</i>
<i>RemoveFromValuesFromRestriction</i> (C_i, OP_i)	—	$\{C_i, OP_i\}$	—	<i>RemoveAssertionObjectProperty</i>
Changements complexes : changements de déplacement				
<i>PullUpClass</i> (C_i, C_p)	—	$(C_i \sqsubseteq C_p) \wedge (\exists(C_x) \in C(O) \cdot (C_i \sqsubseteq C_x))$	$(C_i \sqsubseteq C_x) \wedge (C_p \sqsubseteq C_x)$	<ol style="list-style-type: none"> 1. <i>RemoveDataPropertyAssertion</i> 2. <i>RemoveObjectPropertyAssertion</i>
Changements complexes : changements de fusion				
<i>MergeClasses</i> (C_1, C_2, C_{New})	$\{C_{New}\}$	$\{C_1, C_2\}$	$\{C_{New}\}$	<ol style="list-style-type: none"> 1. <i>AddClass</i> 2. <i>AddIndividual</i> 3. <i>AddSubClass</i> 4. <i>AddEquivalentClasses</i> 5. <i>AddDisjointClasses</i> 6. <i>AddDomainObjectProperty</i> 7. <i>AddRangeObjectProperty</i> 8. <i>RemoveClass</i>

$MergeObjectProperties(OP_1, OP_2, OP_{New})$	$\{OP_{New}\}$	$\{OP_1, C_2\}$	$\{OP_{New}\}$	<ol style="list-style-type: none"> 1. <i>AddObjectProperty</i> 2. <i>AddSubObjectProperty</i> 3. <i>AddEquivalentObjectProperties</i> 4. <i>AddDisjointObjectProperties</i> 5. <i>RemoveObjectProperty</i>
Changements complexes : changements de division				
$SplitClass(C_i, C_{New1}, C_{New2})$	$\{C_{New1}, C_{New2}\}$	$\{C_i\}$	$\{C_{New1}, C_{New2}\}$	<ol style="list-style-type: none"> 1. <i>AddClass</i> 2. <i>AddIndividual</i> 3. <i>AddSubClass</i> 4. <i>AddEquivalentClasses</i> 5. <i>AddDisjointClasses</i> 6. <i>AddDomainObjectProperty</i> 7. <i>AddRangeObjectProperty</i> 8. <i>RemoveClass</i>
$SplitObjectProperties(OP_i, OP_{New1}, OP_{New2})$	$\{OP_{New1}, OP_{New2}\}$	$\{OP_i\}$	$\{OP_{New1}, OP_{New2}\}$	<ol style="list-style-type: none"> 1. <i>AddObjectProperty</i> 2. <i>AddSubObjectProperty</i> 3. <i>AddEquivalentObjectProperties</i> 4. <i>AddDisjointObjectProperties</i> 5. <i>RemoveObjectProperty</i>

TABLE A.1: Formalisation des changements ontologiques avec les grammaires de graphes

Annexe B

Journalisation

Cette annexe présente un extrait d'un fichier de journalisation des changements ontologiques. Le fichier représente formellement le graphe type, le graphe hôte (l'ontologie évoluée) et les différentes règles de réécritures appliquées sur l'ontologie. Ceci permet de garder une trace sur les différents éléments de système de transformations.

```
<?xml version="1.0" encoding="UTF-8"?>
<Document version="1.0">
  <GraphTransformationSystem ID="I1" directed="true" name="GraphTransformationSystem" >
/**** Type Graph ****/
  <Types>
    <NodeType ID="I2" abstract="false" name="Entity%:[NODE]:">
      <AttrType ID="I4" attrname="iri" typename="String" visible="false"/>
      <AttrType ID="I5" attrname="name" typename="String" visible="true"/>
    </NodeType>
    <NodeType ID="I6" abstract="false" name="Class%:[NODE]:">
      <Parent pID="I2"/>
    </NodeType>
    <NodeType ID="I8" abstract="false" name="Property%:[NODE]:">
      <Parent pID="I2"/>
      <AttrType ID="I10" attrname="functional"
        typename="boolean" visible="true"/>
    </NodeType>
    <EdgeType ID="I48" abstract="false" name="domain%:SOLID_LINE:[EDGE]:"/>
    <EdgeType ID="I49" abstract="false" name="range%:SOLID_LINE:[EDGE]:"/>
    <EdgeType ID="I50" abstract="false" name="subclassOf%:[EDGE]:"/>
    <EdgeType ID="I51" abstract="false" name="disjointWith%:[EDGE]:"/>
    <EdgeType ID="I52" abstract="false" name="equivalentTo%:[EDGE]:"/>
    <EdgeType ID="I53" abstract="false" name="complementOf%:[EDGE]:"/>
    <EdgeType ID="I54" abstract="false" name="onProperty%:[EDGE]:"/>
    ....
/***** Host Graph *****/
  <Graph ID="I138" kind="HOST" name="HostGraph">
    <Node ID="I139" type="I24">
      <Attribute constant="true" type="I4">
        <Value> <string>Iiri</string> </Value>
      </Attribute>
      <Attribute constant="true" type="I5">
```

```

        <Value> <string>I</string> </Value>
    </Attribute>
    <NodeLayout X="302" Y="78"/>
    <additionalLayout age="0" force="10" frozen="false" zone="50"/>
</Node>
...
/** Rules */
    <Rule ID="I187" formula="true" name="AddDisjointClasses">
    <Parameter name="x" type="String"/>
/** LHS */
    <Graph ID="I189" kind="LHS" name="LeftOf_AddDisjointClasses">
    <Node ID="I190" type="I6">
    <Attribute constant="true" type="I4">
    <Value> <string>C1iri</string> </Value>
    </Attribute>
    <Attribute constant="true" type="I5">
    <Value> <string>C1</string> </Value>
    </Attribute>
    <NodeLayout X="70" Y="55"/>
    <additionalLayout age="0" force="10" frozen="false" zone="50"/>
    </Node>
    ...
    </Graph>
/** RHS */
    <Graph ID="I198" kind="RHS" name="RightOf_AddDisjointClasses">
    <Node ID="I199" type="I6">
    <Attribute constant="true" type="I4">
    <Value> <string>C1iri</string> </Value>
    </Attribute>
    <Attribute constant="true" type="I5">
    <Value> <string>C1</string> </Value>
    </Attribute>
    <NodeLayout X="137" Y="39"/>
    <additionalLayout age="0" force="10" frozen="false" zone="50"/>
    </Node>
    ...
    <Edge ID="I207" source="I203" target="I199" type="I51">
    <EdgeLayout bendX="0" bendY="0" textOffsetX="0" textOffsetY="-22"/>
    <additionalLayout aktlength="200" force="10" preflength="200"/>
    </Edge>
    <Edge ID="I208" source="I199" target="I203" type="I51">
    <EdgeLayout bendX="0" bendY="0" textOffsetX="0" textOffsetY="-22"/>
    <additionalLayout aktlength="200" force="10" preflength="200"/>
    </Edge>
    </Graph>
    <Morphism comment="Formula: true" name="AddDisjointClasses">
    <Mapping image="I199" orig="I190"/>
    <Mapping image="I203" orig="I194"/>
    </Morphism>
    <ApplCondition>
/** NAC */

```

```
<NAC>
  <Graph ID="I209" kind="NAC" name="NotExist">
    <Node ID="I210" type="I6">
      <Attribute constant="true" type="I4">
        <Value> <string>C1iri</string> </Value>
      </Attribute>
      <Attribute constant="true" type="I5">
        <Value> <string>C1</string> </Value>
      </Attribute>
      <NodeLayout X="66" Y="55"/>
      <additionalLayout age="0" force="10"
        frozen="false" zone="50"/>
    </Node>
    <Node ID="I214" type="I6">
      <Attribute constant="true" type="I4">
        <Value>
          <string>C2iri</string>
        </Value>
      </Attribute>
      <Attribute constant="true" type="I5">
        <Value> <string>C2</string> </Value>
      </Attribute>
      <NodeLayout X="66" Y="171"/>
      <additionalLayout age="0" force="10"
        frozen="false" zone="50"/>
    </Node>
    <Edge ID="I218" source="I210" target="I214" type="I51">
      <EdgeLayout bendX="0" bendY="0"
        textOffsetX="0" textOffsetY="-22"/>
      <additionalLayout aktlength="200" force="10" preflength="200"/>
    </Edge>
  </Graph>
  <Morphism name="NotExist">
    <Mapping image="I210" orig="I190"/>
    <Mapping image="I214" orig="I194"/>
  </Morphism>
</NAC>
<NAC>
  ...
</NAC>
</App1Condition>
<TaggedValue Tag="layer" TagValue="0"/>
<TaggedValue Tag="priority" TagValue="0"/>
</Rule>
</GraphTransformationSystem>
</Document>
```


Bibliographie

- Assmann, U. (1996), How to uniformly specify program analysis and transformation with graph rewrite systems, in *6th International Conference on Compiler Construction (CC 1996)*, pp. 121–135, Springer, Linköping, Sweden.
- Baader, F., I. Horrocks, and U. Sattler (2005), *Mechanizing Mathematical Reasoning*, vol. 2605, chap. Description logics as ontology languages for the semantic web, pp. 228–248, Springer.
- Balasubramanian, D., A. Narayanan, C. van Buskirk, and G. Karsai (2006), The graph rewriting and transformation language : Great, *Electronic Communications of the EASST*, 1.
- Barr, M., and C. Wells (1990), *Category theory for computing science*, vol. 10, Prentice Hall New York.
- Batet, M., D. Sánchez, and A. Valls (2011), An ontology-based measure to compute semantic similarity in biomedicine, *Journal of biomedical informatics*, 44(1), 118–125.
- Baziz, M., N. Aussenac-Gilles, and M. Boughanem (2003), Exploitation des liens sémantiques pour l'expansion de requêtes dans un système de recherche d'information, in *21 ème Congrès INFORMATIQUE des ORGANISATIONS et Systèmes d'Information et de Décision (INFORSID2003)*, pp. 121–134, Nancy, France.
- Bellatreche, L., N. X. Dung, G. Pierra, and D. Hondjack (2006), Contribution of ontology-based data modeling to automatic integration of electronic catalogues within engineering databases, *Computers in Industry*, 57(8), 711–724.
- Birkhoff, G. (1967), *Lattice theory*, vol. 25, American Mathematical Soc.
- Bonfante, G., B. Guillaume, M. Morey, and G. Perrier (2010), Réécriture de graphes de dépendances pour l'interface syntaxe-sémantique, in *Traitement Automatique des Langues Naturelles (TALN 2010)*, Montréal , Canada.
- Borgida, A. (1996), On the relative expressiveness of description logics and predicate logics, *Artificial intelligence*, 82(1), 353–367.

- Bouquet, P., J. Euzenat, E. Franconi, L. Serafini, G. Stamou, and S. Tessaris (2004), D2. 2.1 specification of a common framework for characterizing alignment, *Tech. rep.*, University of Trento.
- Bourigault, D., and N. Aussenac-Gilles (2003), Construction d'ontologies à partir de textes, in *10 ème conférence annuelle sur le Traitement Automatique des Langues, (TALN2003)*, pp. 27–50, Batz-sur-Mer, France.
- Braatz, B., and C. Brandt (2008), Graph transformations for the resource description framework, *Electronic Communications of the EASST*, 10.
- Cafezeiro, I., and E. H. Haeusler (2007), Semantic interoperability via category theory, in *26th international conference on Conceptual Modeling (ER2007)*, pp. 197–202, Australian Computer Society, Auckland, New Zealand.
- Calvanese, D., G. De Giacomo, M. Lenzerini, and D. Nardi (2001), Reasoning in expressive description logics, *Handbook of Automated Reasoning*, 2, 1581–1634.
- Chiticariu, L., P. G. Kolaitis, and L. Popa (2008), Interactive generation of integrated schemas, in *Proceedings international conference on Management of data*, pp. 833–846, ACM, Vancouver, Canada.
- Clarke, E. M., O. Grumberg, and D. Peled (1999), *Model checking*, MIT press.
- Cullot, N., R. Ghawi, and K. Yétongnon (2007), Db2owl : A tool for automatic database-to-ontology mapping., in *Proceedings of the Fifteenth Italian Symposium on Advanced Database Systems, SEBD 2007*, vol. 7, pp. 491–494, Torre Canne, Italy.
- Czarnecki, K., and S. Helsen (2006), Feature-based survey of model transformation approaches, *IBM Systems Journal*, 45(3), 621–645.
- da Costa, S. A., and L. Ribeiro (2012), Verification of graph grammars using a logical approach, *Science of Computer Programming*, 77(4), 480–504.
- d'Aquin, M., P. Doran, E. Motta, and V. A. Tamma (2007), Towards a parametric ontology modularization framework based on graph transformation., in *WoMO*.
- De Lara, J., and H. Vangheluwe (2002), Atom3 : A tool for multi-formalism and meta-modelling, in *Fundamental approaches to software engineering*, pp. 174–188, Springer.
- De Leenheer, P., and T. Mens (2008), Using graph transformation to support collaborative ontology evolution, in *Applications of Graph Transformations with Industrial Relevance*, pp. 44–58, Springer.

- Dershowitz, N., and J.-P. Jouannaud (1989), *Rewrite systems*, Citeseer.
- Dieng, R., O. Corby, F. Gandon, A. Giboin, J. Golebiowska, N. Matta, and M. Ribière (2001), *Méthodes et outils pour la gestion des connaissances : une approche pluridisciplinaire du " Knowledge Management"*, vol. 2, Dunod.
- Djedidi, R. (2009), *Approche d'évolution d'ontologie guidée par des patrons de gestion de changement.*, Ph.D. thesis, Université Paris Sud-Paris XI.
- Djedidi, R., and M.-A. Aufaure (2010), *ONTO-EVO^{al}* an ontology evolution approach guided by pattern modeling and quality evaluation, in *6th International Symposium on Foundations of Information and Knowledge Systems, FOIKS 2010*, pp. 286–305, Springer, Sofia, Bulgaria.
- Do, H.-H., and E. Rahm (2002), *Coma : a system for flexible combination of schema matching approaches*, in *Proceedings of the 28th international conference on Very Large Data Bases*, pp. 610–621, VLDB Endowment.
- Dos Reis, J. C., C. Pruski, and C. Reynaud-Delaître (2015), *State-of-the-art on mapping maintenance and challenges towards a fully automatic approach*, *Expert Systems with Applications*, 42(3), 1465–1478.
- Drewes, F., H.-J. Kreowski, and A. Habel (1999), *Handbook of graph grammars and computing by graph transformation*, chap. Hyperedge Replacement, Graph Grammars., World scientific Singapore.
- Dumais, S. T. (2004), *Latent semantic analysis*, *Annual review of information science and technology*, 38(1), 188–230.
- Dürst, M. J. (2001), *Internationalized resource identifiers : From specification to testing*, in *19th International Unicode Conference, IUC 2001*, San Jose, CA.
- Ehrig, H. (1979), *Introduction to the algebraic theory of graph grammars (a survey)*, in *Graph Grammars and Their Application to Computer Science and Biology*, pp. 1–69, Springer.
- Ehrig, H., U. Montanari, G. Rozenberg, and H. J. Schneider (1996), *Graph Transformations in Computer Science*, Geschäftsstelle Schloss Dagstuhl.
- Engelfriet, J., and G. Rozenberg (1991), *Graph grammars based on node rewriting : An introduction to nlc graph grammars*, in *Graph Grammars and Their Application to Computer Science*, pp. 12–23, Springer.

- Ermel., C., M. Rudolf., and G. Taentzer (1999), The agg approach : Language and environment, in *Handbook of graph grammars and computing by graph transformation*, pp. 551–603, World Scientific Publishing Co., Inc.
- Euzenat, J., and P. Shvaiko (2013), *Ontology Matching, Second Edition*, Springer.
- Fernández-López, M., A. Gómez-Pérez, and N. Juristo (1997), Methontology : from ontological art towards ontological engineering, in *AAAI Symposium on Ontological Engineering*, American Association for Artificial Intelligence.
- Flasiński, M., and J. Jurek (2014), Fundamental methodological issues of syntactic pattern recognition, *Pattern Analysis and Applications*, 17(3), 465–480.
- Flouris, G., Z. Huang, J. Z. Pan, D. Plexousakis, and H. Wache (), Inconsistencies, negations and changes in ontologies, in *the 21st National Conference on Artificial Intelligence, NCAI 2006*, Boston, US.
- Fürst, F., and F. Trichet (2006), Raisonner sur des ontologies lourdes à l’aide de graphes conceptuels., in *24 ème Congrès INFORSID (INFormatique des ORganisations et Systèmes d’Information et de Décision*, pp. 879–894, Hammamet, Tunisie.
- Gagnon, M. (2007), Logique descriptive et owl, *Cours dispensé à l’école polytechnique de Montréal, Canada*.
- Gandon, F. (2002), Ontology engineering : a survey and a return on experience, *Tech. rep.*, INRIA.
- Gandon, F. L., et al. (2008), Graphes rdf et leur manipulation pour la gestion de connaissances, Ph.D. thesis, Université Nice Sophia Antipolis.
- Ganter, B., and R. Wille (2012), *Formal concept analysis : mathematical foundations*, Springer Science & Business Media.
- Garshol, L. M. (2004), Metadata ? thesauri ? taxonomies ? topic maps ! making sense of it all, *Journal of information science*, 30(4), 378–391.
- Genesereth, M. R., R. E. Fikes, et al. (1992), Knowledge interchange format-version 3.0 : Reference manual.
- Giunchiglia, F., and I. Zaihrayeu (2009), Lightweight ontologies, in *Encyclopedia of Database Systems*, pp. 1613–1619, Springer.

- Goldberg, D. E., and J. H. Holland (1988), Genetic algorithms and machine learning, *Machine learning*, 3(2), 95–99.
- Gómez-Pérez, A. (1999), Développement récents en matière de conception, de maintenance et d'utilisation des ontologies, *Terminologies nouvelles*, 19, 9–20.
- Gómez-Pérez, A., M. Fernández, and A. d. Vicente (1996), Towards a method to conceptualize domain ontologies, in *ECAI96 Workshop on Ontological Engineering*, p. 41–51, Budapest, Hongrie.
- Gruber, T. R. (1993), A translation approach to portable ontology specifications, *Knowledge acquisition*, 5(2), 199–220.
- Gueffaz, M. (2012), Scalesem : model checking et web sémantique, Ph.D. thesis, Université de Bourgogne.
- Gueffaz, M., P. Pittet, S. Rampacek, C. Cruz, and C. Nicolle (2012), Inconsistency identification in dynamic ontologies based on model checking, in *8th International Conference on Web Information Systems and Technologies, WEBIST 2012*, pp. 418–421, SciTePress, Porto, Portugal.
- Haase, P., and L. Stojanovic (2005), Consistent evolution of owl ontologies, in *The Semantic Web : Research and Applications, Second European Semantic Web Conference, ESWC 2005*, pp. 182–197, Springer, Heraklion, Crete, Greece.
- Hamdi, F. (2011), Améliorer l'interopérabilité sémantique : applicabilité et utilité de l'alignement d'ontologies., Ph.D. thesis, Paris 11.
- Hartung, M., A. Groß, and E. Rahm (2013), Conto-diff : generation of complex evolution mappings for life science ontologies, *Journal of Biomedical Informatics*, 46(1), 15–32.
- Heckel, R., J. M. Küster, and G. Taentzer (2002), Confluence of typed attributed graph transformation systems, in *Graph Transformation*, pp. 161–176, Springer.
- Hitzler, P., J. Euzenat, M. Krotzsch, L. Serafini, H. Stuckenschmidt, H. Wache, and A. Zimmermann (2006), Integrated view and comparison of alignment semantics, *Tech. rep.*, D2.2.5, AIFB, University of Karlsruhe.
- Hopcroft, J. E., and J. D. Ullman (1969), *Formal languages and their relation to automata*, Addison-Wesley Longman Publishing Co., Inc.
- Javed, M. (2013), Operational change management and change pattern identification for ontology evolution, Ph.D. thesis, Dublin City University.

- Javed, M., Y. M. Abgaz, and C. Pahl (2013), Ontology change management and identification of change patterns, *Journal on Data Semantics*, 2(2-3), 119–143.
- Jaziri, W., N. Sassi, and F. Gargouri (2010), Approach and tool to evolve ontology and maintain its coherence, *International Journal of Metadata, Semantics and Ontologies*, 5(2), 151–166.
- Kalyanpur, A., B. Parsia, E. Sirin, B. C. Grau, and J. Hendler (2006), Swoop : A web ontology editing browser, *Web Semantics : Science, Services and Agents on the World Wide Web*, 4(2), 144–153.
- Kamel, M., and N. Aussenac-Gilles (), Construction automatique d'ontologies à partir de spécifications de bases de données, in *20es Journées Francophones d'Ingénierie des Connaissances, IC 2009*.
- Karsai, G., A. Agrawal, F. Shi, and J. Sprinkle (2003), On the use of graph transformations in the formal specification of computer-based systems, in *Proceedings of IEEE TC-ECBS and IFIP10. 1 Joint Workshop on Formal Specifications of Computer-Based Systems*, pp. 19–27.
- Khalfaoui, K., A. Chaoui, C. Foudil, and E. Kerkouche (2013), Structural validation of software product line variants : A graph transformations based approach, 4(2), 19–31.
- Khattak, A. M., R. Batool, Z. Pervez, A. M. Khan, and S. Lee (2013a), Ontology evolution and challenges, *Journal of Information Science and Engineering*, 29, 851–871.
- Khattak, A. M., K. Latif, and S. Lee (2013b), Change management in evolving web ontologies, *Knowledge-Based Systems*, 37(0), 1–18.
- Kifer, M., and G. Lausen (1989), F-logic : a higher-order language for reasoning about objects, inheritance, and scheme, in *ACM SIGMOD Record*, vol. 18, pp. 134–146, ACM.
- Kifer, M., G. Lausen, and J. Wu (1995), Logical foundations of object-oriented and frame-based languages, *Journal of the ACM (JACM)*, 42(4), 741–843.
- Klein, M. (2001), Combining and relating ontologies : an analysis of problems and solutions, in *Workshop on ontologies and information sharing, IJCAI 2001*, pp. 53–62, Washington, USA.
- Klein, M. (2004), Change management for distributed ontologies, Ph.D. thesis, Vrije Universiteit Amsterdam, Amsterdam, The Netherlands.
- Klein, M., and N. F. Noy (2003), A component-based framework for ontology evolution, in *Proceedings of the Workshop on Ontologies and Distributed Systems, IJCAI, Acapulco, Mexico*.

- Knublauch, H., R. W. Ferguson, N. F. Noy, and M. A. Musen (2004), The protégé owl plugin : An open development environment for semantic web applications, in *The Semantic Web–ISWC 2004*, pp. 229–243, Springer.
- Kotis, K., and G. A. Vouros (2004), The hcone approach to ontology merging, in *The Semantic Web : Research and Applications*, pp. 137–151, Springer.
- Krivine, S., J. Nobécourt, L. Soualmia, F. Cerbah, and C. Duclos (2009), Construction automatique d’ontologie à partir de bases de données relationnelles : application au médicament dans le domaine de la pharmacovigilance, in *20es Journées Francophones d’Ingénierie des Connaissances, IC 2009*, pp. 73–84.
- Levenshtein, V. I. (1966), Binary codes capable of correcting deletions, insertions and reversals, in *Soviet physics doklady*, vol. 10, pp. 707–710.
- Li, G., Z. Luo, and J. Shao (2010), Multi-mapping based ontology merging system design, in *2nd International Conference on Advanced Computer Control (ICACC)*, vol. 2, pp. 5–11, IEEE.
- Liu, L., P. Zhang, R. Fan, R. Zhang, and H. Yang (2014), Modeling ontology evolution with setpi, *Information Sciences*, 255, 155–169.
- Lladós, J., and G. Sánchez (2003), Symbol recognition using graphs, in *Proceedings International Conference on Image Processing, ICIP 2003*, pp. 49–52, IEEE.
- Löwe, M. (1993), Algebraic approach to single-pushout graph transformation, *Theoretical Computer Science*, 109(1), 181–224.
- Luong, P.-H., and R. Dieng-Kuntz (2007), A rule-based approach for semantic annotation evolution, *Computational Intelligence*, 23(3), 320–338.
- Maedche, A., and S. Staab (2000), Semi-automatic engineering of ontologies from text, in *Proceedings of the 12th international conference on software engineering and knowledge engineering*, pp. 231–239, Chicago, IL, USA.
- Mahfoudh, M., and W. Jaziri (2013), Approche de couplage de bd et d’ontologie pour l’aide à la décision sémantique : contribution pour la satisfaction des requêtes sql et sparql, *TSI. Technique et science informatiques*, 32(7-8), 863–889.
- Mahfoudh, M., G. Forestier, L. Thiry, and M. Hassenforder (2013a), Consistent ontologies evolution using graph grammars, in *Knowledge Science, Engineering and Management*, pp. 64–75, Springer, Dalian, China.

- Mahfoudh, M., L. Thiry, G. Forestier, and M. Hassenforder (2013b), Adaptation consistante d'ontologies à l'aide des grammaires de graphe, in *Conférence d'Ingénierie des Connaissances*, Lille, France.
- Mahfoudh, M., G. Forestier, L. Thiry, and M. Hassenforder (2014a), Approche formelle de fusion d'ontologies à l'aide des grammaires de graphes typés, in *14èmes Journées Francophones Extraction et Gestion des Connaissances, EGC 2014*, pp. 565–568, Rennes, France.
- Mahfoudh, M., G. Forestier, L. Thiry, and M. Hassenforder (2014b), Comment fusionner des ontologies avec la réécriture de graphes?, in *Journées Francophones sur les ontologies, JFO 2014*, pp. 89–100, Hammamet, Tunisie.
- Mahfoudh, M., L. Thiry, G. Forestier, and M. Hassenforder (2014c), Algebraic graph transformations for merging ontologies, in *Model and Data Engineering, MEDI 2014*, pp. 154–168, Springer, Larnaca, Chypre.
- Mahfoudh, M., G. Forestier, L. Thiry, and M. Hassenforder (2015a), Algebraic graph transformations for formalizing ontology changes and evolving ontologies, *Knowledge-Based Systems*, 73, 212–226.
- Mahfoudh, M., L. Thiry, G. Forestier, and M. Hassenforder (2015b), Une nouvelle formalisation des changements ontologiques composés et complexes, in *15èmes Journées Francophones Extraction et Gestion des Connaissances, EGC 2015*, pp. 263–274, RNTI (E28), Luxembourg, Luxembourg.
- McBride, B. (2004), The resource description framework (rdf) and its vocabulary description language rdfs, in *Handbook on ontologies*, pp. 51–65, Springer.
- McGuinness, D. L., R. Fikes, L. A. Stein, and J. A. Hendler (2003), Daml-ont : An ontology language for the semantic web., in *Spinning the Semantic Web*, pp. 65–93.
- McGuinness, D. L., F. Van Harmelen, et al. (2004), Owl web ontology language overview, *W3C recommendation*, 10(10), 2004.
- Mens, T., P. Van Gorp, D. Varró, and G. Karsai (2006), Applying a model transformation taxonomy to graph transformation technology, *Electronic Notes in Theoretical Computer Science*, 152, 143–159.
- Mens, T., G. Taentzer, and O. Runge (2007), Analysing refactoring dependencies using graph transformation, *Software & Systems Modeling*, 6(3), 269–285.

- Mhiri, M. B. A., F. Gargouri, and D. Benslimane (2006), Détermination automatique des relations sémantiques entre les concepts d'une ontologie, in *24^{ème} Congrès INFORSID (INFormatique des ORganisations et Systèmes d'Information et de Décision)*, pp. 627–642, Hammamet, Tunisie.
- Miles, A., and J. R. Pérez-Agüera (2007), Skos : Simple knowledge organisation for the web, *Cataloging & Classification Quarterly*, 43(3-4), 69–83.
- Miller, E. (1998), An introduction to the resource description framework, *Bulletin of the American Society for Information Science and Technology*, 25(1), 15–19.
- Miller, G. A. (1995), Wordnet : A lexical database for english, *Communications of the ACM*, 38(11), 39–41.
- Milner, R. (1999), *Communicating and mobile systems : the pi calculus*, Cambridge university press.
- Minsky, M. (1975), A framework for representing knowledge, *The psychology of computer vision*, pp. 211–277.
- Motik, B., R. Shearer, and I. Horrocks (2009), Hypertableau reasoning for description logics, *Journal of Artificial Intelligence Research*, 36(1), 165–228.
- Neches, R., R. E. Fikes, T. Finin, T. Gruber, R. Patil, T. Senator, and W. R. Swartout (1991), Enabling technology for knowledge sharing, *AI magazine*, 12(3), 36.
- Nickel, U., J. Niere, and A. Zündorf (2000), The fujaba environment, in *Proceedings of the 22nd international conference on Software engineering*, pp. 742–745, ACM.
- Noy, N. F., and C. D. Hafner (1997), The state of the art in ontology design : A survey and comparative review, *AI magazine*, 18(3), 53.
- Noy, N. F., and M. A. Musen (2000), Algorithm and tool for automated ontology merging and alignment, in *17th National Conference on Artificial Intelligence (AAAI)*, pp. 450–455, AAAI Press / The MIT Press, Georgia, USA.
- Noy, N. F., and M. A. Musen (2001), Anchor-prompt : Using non-local context for semantic matching, in *Proceedings of the workshop on ontologies and information sharing at the international joint conference on artificial intelligence (IJCAI)*, pp. 63–70, Washington, USA.
- Noy, N. F., and M. A. Musen (2003), The prompt suite : interactive tools for ontology merging and mapping, *International Journal of Human-Computer Studies*, 59(6), 983–1024.

- Pavel, S., and J. Euzenat (2013), Ontology matching : State of the art and future challenges, *IEEE Transactions on Knowledge and Data Engineering*, 25(1), 158–176.
- Peukert, E., S. Massmann, and K. Koenig (2010), Comparing similarity combination methods for schema matching., in *GI Jahrestagung (1)*, pp. 692–701, Citeseer.
- Pittet, P. (2014), Ontoversiongraph : A change management methodology dedicated to formal ontologies and their user views in a collaborative context : application to shoin(d) ontologies, Ph.D. thesis, Université de Bourgogne.
- Pittet, P., C. Cruz, and C. Nicolle (2014), An ontology change management approach for facility management, *Computers in Industry*, 65(9), 1301–1315.
- Plump, D. (1994), Critical pairs in term graph rewriting, in *Mathematical Foundations of Computer Science 1994*, pp. 556–566, Springer.
- Pnueli, A. (1977), The temporal logic of programs, in *18th Annual Symposium on Foundations of Computer Science, 1977*, pp. 46–57, IEEE.
- Raunich, S., and E. Rahm (2012), Towards a benchmark for ontology merging, in *On the Move to Meaningful Internet Systems : OTM 2012 Workshops*, pp. 124–133, Springer.
- Raunich, S., and E. Rahm (2014), Target-driven merging of taxonomies with atom, *Information Systems*, 42, 1–14.
- Rebout, M. (2008), Une approche catégorique unifiée pour la réécriture de graphes attribués, Ph.D. thesis, Université Paul Sabatier-Toulouse III.
- Royer, K., L. Bellatreche, and S. Jean (2014), Combining domain and business ontologies in a modular construction method : EDF study case, in *37th International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2014*, pp. 1452–1457, Adriatic Coast, Croatia.
- Rozenberg, G. (1999), *Handbook of graph grammars and computing by graph transformation*, vol. 1, World Scientific.
- Schneider, K. (1997), Application of graph grammars in an educational software engineering game : A case study in pragmatic adoption, *International Journal of Software Engineering and Knowledge Engineering*, 7(04), 401–429.
- Schürr, A. (1995), Specification of graph translators with triple graph grammars, in *Graph-Theoretic Concepts in Computer Science*, pp. 151–163, Springer.

- Schürr, A., A. J. Winter, and A. Zündorf (1999), The progres approach : Language and environment, p. 487–550.
- Sellami, Z. (2012), Gestion dynamique d'ontologies à partir de textes par systèmes multi-agents adaptatifs, Ph.D. thesis, Université de Toulouse, Université Toulouse III-Paul Sabatier.
- Sellami, Z., V. Camps, and N. Aussenac-Gilles (2013), Dynamo-mas : a multi-agent system for ontology evolution from text, *Journal on Data Semantics*, 2(2-3), 145–161.
- Shaban-Nejad, A., and V. Haarslev (2015), Managing changes in distributed biomedical ontologies using hierarchical distributed graph transformation, *International Journal of Data Mining and Bioinformatics*, 11(1), 53–83.
- Sirin, E., B. Parsia, B. Grau, A. Kalyanpur, and Y. Katz (2007), Pellet : A practical owl-dl reasoner, *Web Semantics : science, services and agents on the World Wide Web*, 5(2), 51–53.
- Spivey, J. M., and J. Abrial (1992), *The Z notation*, Prentice Hall Hemel Hempstead.
- Stoilos, G., G. Stamou, and S. Kollias (2005), A string metric for ontology alignment, in *The Semantic Web–ISWC 2005*, pp. 624–637, Springer, Galway, Ireland.
- Stojanovic, L. (2004), Methods and tools for ontology evolution, Ph.D. thesis, University of Karlsruhe, Germany.
- Stojanovic, N., L. Stojanovic, and S. Handschuh (2002), Evolution in the ontology-based knowledge management system, in *Proceedings of the European Conference on Information Systems-ECIS*.
- Stumme, G., and A. Maedche (2001), Fca-merge : Bottom-up merging of ontologies, in *Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001*, pp. 225–230, Washington, USA.
- Thiry, L., M. Mahfoudh, and M. Hassenforder (2014), A functional inference system for the web, *International Journal of Web Applications*, 6, 1–13.
- Touhami, R., P. Buche, J. Dibie-Barthélemy, and L. Ibănescu (2011), An ontological and terminological resource for n-ary relation annotation in web data tables, in *On the Move to Meaningful Internet Systems : OTM 2011*, pp. 662–679, Springer, Crete, Greece.
- Tsarkov, D., and I. Horrocks (2006), Fact++ description logic reasoner : System description, in *Third International Joint Conference on Automated Reasoning, IJCAR 2006*, pp. 292–297, Springer, WA, USA.

- Uschold, M., and M. Gruninger (1996), *Ontologies : Principles, methods and applications*, *The knowledge engineering review*, 11(02), 93–136.
- Uschold, M., and M. King (1995), *Towards a methodology for building ontologies*, Citeseer.
- Vanlande, R., C. Nicolle, and C. Cruz (2008), *Ifc and building lifecycle management*, *Automation in Construction*, 18(1), 70–78.
- Varró, D., and A. Pataricza (2003), *Vpm : A visual, precise and multilevel metamodeling framework for describing mathematical domains and uml (the mathematics of metamodeling is metamodeling mathematics)*, *Software and Systems Modeling*, 2(3), 187–210.
- Varró, D., and A. Pataricza (2004), *Generic and meta-transformations for model transformation engineering*, *2004-The Unified Modeling Language. Modelling Languages and Applications*, pp. 290–304.
- Welty, C., and N. Guarino (2001), *Supporting ontological analysis of taxonomic relationships*, *Data & Knowledge Engineering*, 39(1), 51–74.
- Zimmermann, A., M. Krotzsch, J. Euzenat, and P. Hitzler (2006), *Formalizing ontology alignment and its operations with category theory*, *Frontiers in Artificial Intelligence and Applications*, 150, 277–288.
- Zouaq, A., and R. Nkambou (2008), *Building domain ontologies from text for educational purposes*, *Learning Technologies, IEEE Transactions on*, 1(1), 49–62.