Une nouvelle formalisation des changements ontologiques composés et complexes

Mariem Mahfoudh*, Laurent Thiry*, Germain Forestier*, Michel Hassenforder*

*MIPS EA 2332, Université de Haute Alsace
12 rue des Frères Lumière, 68093 Mulhouse, France
{mariem.mahfoudh, laurent.thiry, germain.forestier, michel.hassenforder}@uha.fr

Résumé. L'évolution d'une ontologie est un processus indispensable dans son cycle de vie. Elle est exprimée et définie par des changements ontologiques de différents types : élémentaires, composés et complexes. Les changements complexes et composés 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. Pour adresser cette problématique, cet article présente une nouvelle formalisation des changements ontologiques composés et complexes basée sur les grammaires de graphes typés. Cette formalisation s'appuie sur l'approche algébrique Simple Pushout (SPO) de transformation de graphes et possède deux principaux avantages : (1) fournir une nouvelle formalisation permettant de contrôler les transformations de graphes et éviter les incohérences d'une manière a priori, (2) simplifier la définition des changements composés et complexes en réduisant le nombre de changements élémentaires nécessaires à leur application.

1 Introduction

L'évolution d'ontologie est un sujet posé avec l'apparition des méthodologies de construction d'ontologies. Il s'est avéré indispensable de penser à maintenir et faire évoluer les ontologies, après leur construction, afin d'assurer leur réutilisation et leur continuité. Ce besoin s'est rapidement développé avec la prolifération des ontologies et leur large utilisation. À titre d'exemple, depuis Janvier 2010, 59 versions de l'ontologie *Gene Ontology* [1](une des plus fameuses ontologies) ont été publiées à raison d'une version par mois. Ainsi, afin de définir et gérer le processus d'évolution, plusieurs méthodologies ont été proposées dans la littérature (Klein, 2004; Stojanovic, 2004; Djedidi et Aufaure, 2010; Khattak et al., 2013). Les premiers travaux ont pensé à définir ce qu'est une évolution d'ontologie. D'où la définition proposée par (Stojanovic, 2004) : *"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"*. Cette définition a ouvert le débat sur la signification des changements ontologiques,

1. geneontology.org/ontology-archive

leurs formalisations et leurs types (Klein, 2004; Stojanovic, 2004). Ainsi, un changement ontologique est une modification d'une ou plusieurs entités ontologiques (classe, propriété, axiome, individus, etc.). Il peut 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*. Il peut viser également l'ajout d'individus et on parle alors du *peuplement d'ontologie*. Les changements ontologiques sont souvent de trois types (Stojanovic, 2004) : 1) les *changements élémentaires* qui représentent une opération primitive et non décomposable qui affecte une seule entité ontologique (ex. renommer une classe) ; 2) les *changements composés (composites)* qui affectent une entité ontologique et ses voisins (ex. suppression d'une classe) ; 3) les *changements complexes* qui expriment un enchainement de plusieurs changements élémentaires et/ou composés (ex. fusion de classes). En effet, les changements composés et complexes sont des changements utiles et demandés par l'utilisateur. Ils englobent plusieurs modifications en une seule opération, ce qui lui permet d'adapter son ontologie d'une manière plus facile sans se perdre dans les détails des changements élémentaires (Klein et Noy, 2003). Cependant, la définition et la formalisation de ces changements sont des tâches non triviales comme leur application peut affecter la cohérence de l'ontologie. Deux types de cohérence sont généralement distingués dans la littérature : 1) la *cohérence conceptuelle* qui se réfère aux règles structurelles et contraintes du langage de représentation de l'ontologie (ex. inexistence de concepts isolés) ; 2) la *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 des relations contradictoires entre deux concepts).
En effet, la préservation de la consistance de l'ontologie et la résolution des incohérences résultantes de l'application des changements ontologiques sont encore des problématiques insuffisamment étudiées. Ainsi, nous proposons dans cet article une nouvelle formalisation des changements ontologiques composés et complexes permettant : 1) d'éviter les inconsistances d'une manière a priori en utilisant les concepts des grammaires de graphes typés ; 2) de réduire le nombre de changements élémentaires constituant les changements composés/complexes. Les changements étudiés traitent à la fois le niveau structurel de l'ontologie (l'enrichissement de l'ontologie) et aussi le niveau assertionnel (le peuplement d'ontologie).

Le reste de l'article sera organisé comme suit : la section 2 présente un tour d'horizon sur les principales approches d'évolution d'ontologie. La section 3 introduit les concepts de base des grammaires de graphes. La section 4 propose une nouvelle formalisation des changements ontologiques composés et complexes. Enfin, une conclusion synthétise le travail présenté et donne les perspectives envisagées.

2 État de l'art

De nombreuses approches ont été proposées dans la littérature pour définir et implémenter le processus d'évolution d'ontologies. Le Tableau 1 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 et Dieng-Kuntz, 2007), OWL (Klein, 2004; Djedidi et 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 intéressés à l'étude des changements élémentaires (Mahfoudh

et al., 2013). D'autres ont traité également les changements composés et complexes (Djedidi et Aufaure, 2010; Javed et al., 2013; Liu et al., 2014). Des travaux se sont focalisés sur l'enrichissement d'ontologies (Klein, 2004). D'autres ont étudié aussi le peuplement d'ontologies (Luong et Dieng-Kuntz, 2007; Djedidi et Aufaure, 2010; Mahfoudh et al., 2013). La résolution des inconsistances est encore insuffisamment étudié. En effet, certaines approches ont ignoré cet axe comme 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 et Aufaure, 2010; Luong et Dieng-Kuntz, 2007; Javed et al., 2013). Cependant, les approches proposées admettant un processus a postériori 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. Afin d'éviter l'utilisation d'un raisonneur externe, Mahfoudh et al. (2013) ont proposé une approche a priori de résolution des inconsistances basée sur les grammaires de graphes typés. Cependant, le travail ne présente que des changements élémentaires. Dans cet article, nous proposons d'utiliser le même formalisme pour définir des changements composés et complexes.

Approches	**Langages d'ontologies**	**Implémentation**	**Gestion des inconsistances**	**Spécificités**
(Stojanovic, 2004)	KAON	Framework KAON	- Identification de certaines inconsistances. - Stratégies proposées à l'ontologiste pour résoudre les inconsistances.	- 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.
(Klein, 2004)	OWL	OntoView, PROMPTdiff	- Identification et résolution des inconsistances.	- 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.
(Djedidi et Aufaure, 2010)	OWL DL	Prototype Onto-EVO^{A}L	- Identification des inconsistances en utilisant le raisonneur Pellet.	- Approche basée sur les patrons de conception. - Évaluation de la qualité de l'ontologie évoluée. - Approche nécessitant des activités lourdes.
(Gueffaz et al., 2012)	OWL DL	Prototype	- Identification des inconsistances en utilisant le checker NuSMV.	- 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.
(Hartung et al., 2013)	OBO (Open Biomedical Ontologies) et RDF	L'outil Contodiff, L'application web OnEX	—	- 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).
(Khattak et al., 2013)	RDFS et OWL	Plugin Protégé	- Gérer les inconsistances en utilisant l'API KAON (Stojanovic, 2004).	- Approche de gestion des versions d'ontologies.
(Liu et al., 2014)	OWL	—	—	- Proposition d'un nouveau formalisme, le SetPi-calcul, pour modéliser l'évolution d'ontologies. - Formalisation des changements élémentaires, composés et complexes.

(Mahfoudh et al., 2013)	OWL et GGX	Implémentation avec l'outil AGG	- Éviter les inconsistances à l'aide des règles de réécriture de graphes.	- Approche a priori de résolutions des inconsistances. - Proposition d'un nouveau formalisme, les grammaires de graphes typés. - Traitement des changements élémentaires.

TAB. 1: APPROCHES D'ÉVOLUTION D'ONTOLOGIE.

3 Grammaires de Graphes Typés

Les grammaires de graphes, également appelées réécriture de graphes, sont un formalisme mathématique pour représenter et gérer les graphes. Elles permettent la modification de graphes via des règles de réécriture tout en précisant quand et comment faire les changements. Grâce aux concepts et outils qu'elles proposent, les grammaires de graphes sont utilisées dans plusieurs branches de l'informatique, comme par exemple la modélisation des systèmes logiciels et la théorie des langages formels (Ehrig et al., 1996). Elles ont récemment été introduites dans le domaine des ontologies, ce qui a donné naissance à des travaux traitant de la formalisation des ontologies modulaires (d'Aquin et al., 2007), la représentation des graphes RDF (Resource Description Framework) (Braatz et Brandt, 2010), la fusion d'ontologies (Mahfoudh et al., 2014a), etc. Dans ce qui suit, nous dressons un tour d'horizon des définitions de base concernant les fondements théoriques de la réécriture de graphes.

Graphe. Un graphe $G(N, E)$ est une structure composée par un ensemble de nœuds (N), d'arêtes (E) et d'une application $s : E \rightarrow N \times N$ qui attache les nœuds source/destination à chaque arête.

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 $val : A \rightarrow V$. Ainsi, chaque nœud ou arête peut avoir un ensemble d'attributs ($\mathcal{P}(A)$) dont les valeurs seront données par val.

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 $g : E \rightarrow E'$. Un morphisme doit préserver la structure, c.à.d si $e = (s, t)$ et $g(e) = e' = (s', t')$ alors $s' = f(s)$ et $t' = f(t)$.

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

Grammaires de graphes typés. Une grammaire de graphe typé est une structure mathématique définie par $TGG = (G, TG, P)$ avec :

- G est un graphe initial, appelé aussi graphe hôte ;
- TG est un graphe type précisant le type de l'information représentée dans le graphe hôte (type des nœuds et des arêtes) ;
- P un ensemble de règles de réécriture, appelées aussi règles de production ou de transformations de graphes. Une règle de réécriture r est une paire de graphes pattern (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 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 peuvent également avoir des conditions supplémentaires appelées NAC (Negative Application Conditions). Ce sont des graphes pattern définissant des conditions ne devant pas être vérifiées pour que la règle de réécriture puisse être appliquée.

La transformation de graphe consiste ainsi à définir comment un graphe G peut être transformé en un nouveau graphe G'. Cette transformation peut être réalisée selon deux types d'approches (Rozenberg, 1999) : les approches ensemblistes (Node replacement, Edge replacement, etc.) et les approches algébriques. Dans ce travail, nous utilisons les approches algébriques basées sur le concept de *pushout* de la théorie des catégories (Ehrig et al., 1973).

Théorie des catégories. Une catégorie 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 $(\circ) : M \times M \rightarrow M$; 4) un morphisme identité pour chaque objet $id : O \rightarrow O$. La loi de composition doit être associative et avoir l'identité comme élément neutre. Par exemple, dans la catégorie *Graph*, O représente l'ensemble des graphes et M les morphismes de graphes. Les éléments présentés dans la suite s'appuient sur cette catégorie.

Pushout. Soient trois objets de la catégorie des graphes : G_1, G_2 et G_3 et deux morphismes $f : G_1 \rightarrow G_2$ et $g : G_1 \rightarrow G_3$. Le pushout de G_2 et G_3 consiste à : 1) un objet G_4 et deux morphismes $f' : G_2 \rightarrow G_4$ et $g' : G_3 \rightarrow G_4$ avec $f' \circ f = g' \circ g$; 2) pour tout morphisme $f'' : G_2 \rightarrow X$ et $g'' : G_3 \rightarrow X$ tel que $f \circ f'' = g \circ g''$, il y a un seul morphisme $k : G_4 \rightarrow X$ tel que $f' \circ k = f''$ et $g' \circ k = g''$.

A partir de là, deux variantes sont proposées pour la réécriture des graphes : le *Simple pushout SPO* (Löwe, 1993) et le *Double poushout DPO* (Ehrig, 1979). Dans ce travail, seule l'approche SPO a été considérée car elle se voit plus générale et permet l'application des différents changements ontologiques (Mahfoudh et al., 2014b). Ainsi, appliquer une règle de réécriture à un graphe initial G, selon la méthode SPO, revient à : 1) trouver un morphisme (m) permettant d'identifier un sous-graphe de graphe G qui correspond $(match)$ avec la partie LHS (m : $LHS \rightarrow G$) ; 2) appliquer la règle de réécriture sur le sous-graphe en le remplaçant par $m(RHS)$ et supprimant les arêtes suspendues, i.e. les arêtes qui ont une extrémité non liée à un nœud. Ainsi, d'une manière générale, nous avons $SPO(G, LHS, RHS) = G'$.

4 Formalisation des changements ontologiques

4.1 Modèle de transformation de graphes

Afin de représenter les ontologies avec le formalisme de grammaires de graphes, nous considérons une ontologie comme un graphe hôte G possédant une relation de typage avec le graphe type (TG), où TG représente le méta-modèle de l'ontologie. Pour être conforme aux standards, c'est OWL qui a été retenu comme méta-modèle d'ontologies. Ainsi, les types des nœuds considérés sont :

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

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

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

Les changements ontologiques sont formalisés par un ensemble de règles de réécriture :

$$r_i = (NAC_i, LHS_i, RHS_i, CHD_i) \text{ avec}$$
$$i \in \{AddClass, RemoveDataProperty, RenameIndividual, ...\}.$$

Dans cette définition étendue, CHD correspond à l'ensemble des changements dérivés ajoutés à un changement ontologique pour corriger ses éventuelles inconsistances.
La Figure 1 montre une représentation et une application de la règle de réécriture du changement ontologique $AddIndividual$. Elle permet d'ajouter un individu "Pascal" tout en spécifiant son type, la classe "Person". La règle assure, grâce au NAC, la non redondance de données, i.e. elle empêche l'application du changement dans le cas où l'individu existe déjà dans l'ontologie.

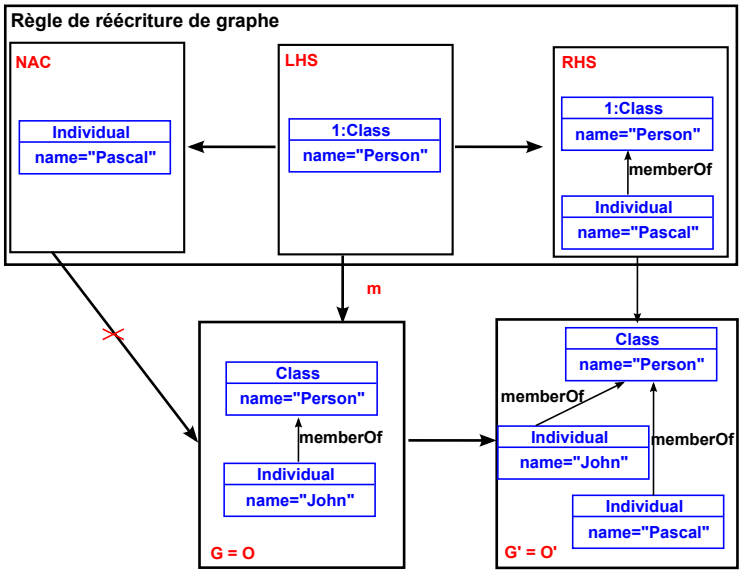


FIG. 1 – *Règle de réécriture du changement $AddIndividual$.*

4.2 Formalisation des changements ontologiques

Cette section présente notre formalisation des changements ontologiques composés et complexes par les grammaires de graphes typés. Avant de détailler cette formalisation, une brève introduction des changements élémentaires est indispensable pour mieux comprendre le reste de l'article. Ainsi, les changements élémentaires englobent les changements de renommage, l'ajout et la suppression de certains concepts. Ils n'affectent qu'une seule entité ontologique bien qu'ils dépendent d'autres entités. Le Tableau 2 présente quelques changements élémentaires adressés dans notre travail et les concepts dont ils sont dépendant. À noter que les $NACs$ des règles de réécriture sont déduites à partir de ces interdépendances. Un exemple du changement élémentaire $AddIndividual$ est déjà présenté dans la section 4.1.

	Class	Individual	ObjectProperty	DataProperty	DataType	EquivalentClass	DisjointClass	SubClass	EquivalentProperty	DisjointProperty	SubProperty	FunctionalProperty	CardinalityRestriction	AllValuesFromRestriction
RenameClass	✓													
AddDataProperty	✓			✓	✓									
AddDisjointClasses	✓	✓				✓	✓	✓						
AddDataPropertyAssertion		✓		✓								✓		
AddSubObjectProperty			✓							✓	✓			
AddCardinalityRestriction	✓		✓										✓	
AddAllValuesFromRestriction	✓		✓											✓
RemoveIndividual		✓												
RemoveEquivalentObjectProperties			✓						✓					

TAB. 2: MATRICE DE DÉPENDANCE ENTRE LES CHANGEMENTS ÉLÉMENTAIRES ET LES ENTITÉS ONTOLOGIQUES.

4.2.1 Changements ontologiques composés

Les changements ontologiques composés, appelés aussi composites, affectent une entité ontologique et ses voisins. Ils sont alors formés par plusieurs règles de réécriture : une règle présentant le changement souhaité par l'utilisateur (changement principal) et les autres règles présentant les changements dérivés (CHD) ajoutés pour préserver la consistance de l'ontologie. En effet, l'ordre des règles de réécriture est primordial dans la plupart des changements. Le Tableau 3 présente l'interdépendance entre ces changements 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.

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

TAB. 3: MATRICE DE DÉPENDANCE ENTRE LES CHANGEMENTS ONTOLOGIQUES COMPOSÉS.

Ainsi, le changement ontologique $RemoveCardinalityRestriction(C, OP)$ permet de supprimer une $CardinalityRestriction$ définie sur une classe C et une objectProperty 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.

Le changement ontologique $RemoveObjectProperty(OP)$ supprime une objectProperty OP et toutes ses dépendances de l'ontologie. La Figure 2 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.

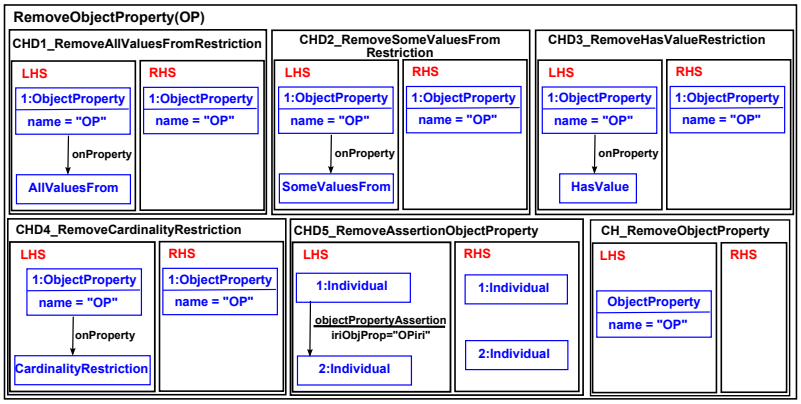


FIG. 2 – *La règle de réécriture du changement ontologique* $RemoveObjectProperty$.

4.2.2 Changements ontologiques complexes

Le Tableau 4 présente l'ensemble des changements complexes abordé dans ce travail et les changements dont ils sont composés.

Comme exemples de changements complexes, nous présentons les changements $PullUpClass$, $MergeClass$ et $SplitClass$. 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 3 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.

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

TAB. 4: MATRICE DE DÉPENDANCE DES CHANGEMENTS ONTOLOGIQUES COMPLEXES.

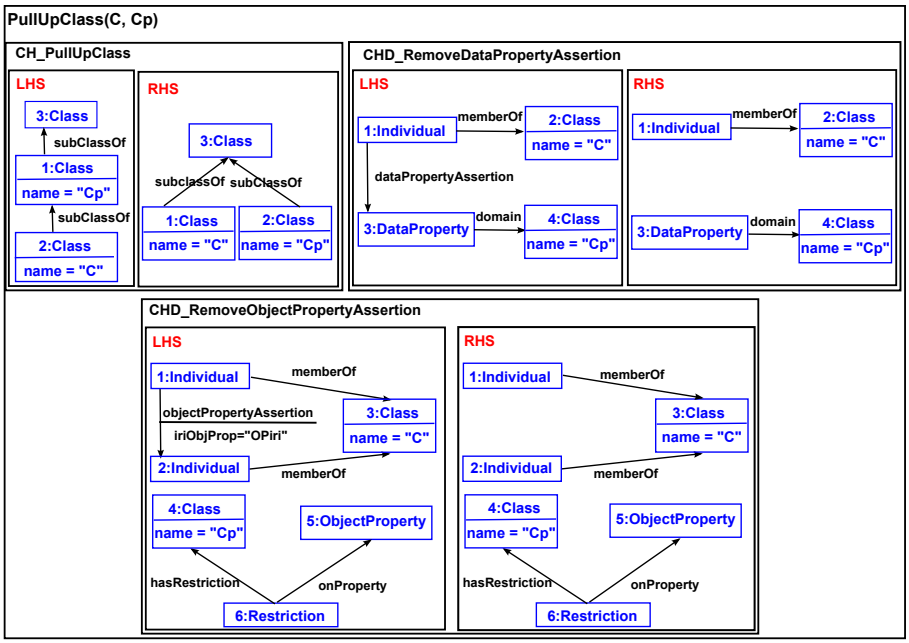


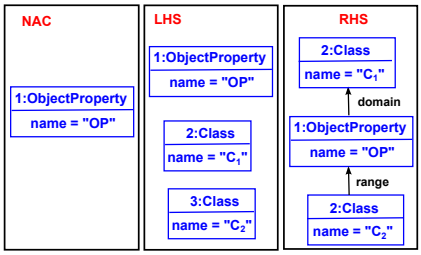
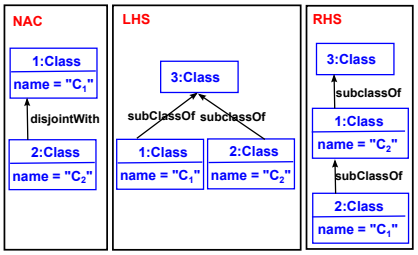
FIG. 3 – *La règle de réécriture du changement* $PullUpClass$.

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 $AddClass(C_{New})$, $RemoveClass(C_1)$ 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 le processus pour C_2, 3) $\forall C_i \in C(O) \cdot C_i \equiv C_1$ appliquer $AddEquivalentClasses(C_i, C_{New})$, 4) répéter le processus pour C_2, etc.

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 alors 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}.

Discussion : Le formalisme des grammaires de graphes offre une représentation simple des changements ontologiques. Le Tableau 5 montre deux exemples de changements $AddObjectProperty$ et $PullDownClass$, représentés à la fois par le formalisme proposé et le travail de (Djedidi et Aufaure, 2010). Dans Djedidi et Aufaure (2010), ces changements sont considérés, respectivement, comme composés et complexes. Le premier changement est composé par trois changements élémentaires et le deuxième par deux. De plus, ils nécessitent, comme tous les autres changements ontologiques, 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 Negatives Applications Conditions (NAC).

Changement ontologique	**(Djedidi et Aufaure, 2010)**	**Formalisme proposé**
$AddObjectProperty(OP, C_1, C_2)$	Le changement est composé de trois changements élémentaires : 1. $AddObjectProperty(OP)$, 2. $AddDomain(OP, C_1)$ 3. $AddRange(OP, C_2)$.	- Le changement est formalisé par une seule règle de réécriture et évite la non-redondance de données. 
$PullDownClass(C_1, C_2)$: descend une classe (C_1) de sa hiérarchie de classes et l'attacher comme sub-classe de sa précédente classe sœur (C_2).	Le changement est composé par deux changements élémentaires : 1. $AddSubClass(C_1, C_2)$ 2. $RemoveSubClass(C_1, C_p)$avec la classe C_p est la super-classe de C_1 et C_2.	- Le changement est formalisé par une seule règle de réécriture et évite la contradiction des axiomes. 

TAB. 5: FORMALISATION DES CHANGEMENTS ONTOLOGIQUES SELON (DJEDIDI ET AUFAURE, 2010) ET L'APPROCHE PROPOSÉE.

5 Conclusion

Nous avons présenté dans cet article une nouvelle formalisation des changements ontologiques composés et complexes basée sur les grammaires de graphes typés et l'approche algébrique Simple Pushout (SPO) de transformation de graphes. L'utilisation de l'approche SPO offre plusieurs avantages. En particulier, elle permet de définir simplement et formellement les règles de réécriture correspondant aux changements ontologiques. Elle assure le contrôle des transformations de graphes en évitant les incohérences d'une manière a priori. De plus, elle réduit le nombre de changements élémentaires nécessaires pour appliquer les changements complexes et composés. À noter que la formalisation des changements a été implémentée à l'aide de l'outil AGG (Attributed Graph Grammar) et testée sur des ontologies de taille réduite. En effet, l'étape la plus coûteuse en temps et en ressources est la reconnaissance du graphe LHS à partir du graphe hôte. Vu que la plupart des changements possèdent des LHS de taille réduite, il s'est avéré que le temps d'exécution est assez limité. À titre d'exemple, pour un graphe d'une ontologie composé de 21 nœuds, l'exécution du changement complexe $SplitClass(C, C_{New1}, C_{New2})$ a pris seulement 700 millisecondes (avec un LHS composé de 37 nœuds). Pour mieux évaluer la performance de notre approche, nous travaillons actuellement sur l'évaluation de l'influence de la taille de LHS sur les ontologies de grande taille.

Références

Braatz, B. et C. Brandt (2010). How to modify on the semantic web ? - A web application architecture for algebraic graph transformations on RDF. In *10th International Conference on Web Engineering, ICWE 2010 Workshops*, pp. 187–198. Springer.

d'Aquin, M., P. Doran, E. Motta, et V. A. Tamma (2007). Towards a parametric ontology modularization framework based on graph transformation. In *Workshop on Modular Ontologies*.

Djedidi, R. et M.-A. Aufaure (2010). *ONTO-EVO^{a}l* an ontology evolution approach guided by pattern modeling and quality evaluation. In *Foundations of Information and Knowledge Systems*, pp. 286–305. Springer.

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, et H. J. Schneider (1996). *Graph Transformations in Computer Science*. Geschäftsstelle Schloss Dagstuhl.

Ehrig, H., M. Pfender, et H. J. Schneider (1973). Graph-grammars : An algebraic approach. In *Switching and Automata Theory, 1973. SWAT'08. IEEE Conference Record of 14th Annual Symposium on*, pp. 167–180. IEEE.

Gueffaz, M., P. Pittet, S. Rampacek, C. Cruz, et C. Nicolle (2012). Inconsistency identification in dynamic ontologies based on model checking. In *International Conference on Web Information Systems and Technologies*, pp. 418–421. SciTePress.

Hartung, M., A. Groß, et E. Rahm (2013). Conto–diff : generation of complex evolution mappings for life science ontologies. *Journal of Biomedical Informatics 46*(1), 15–32.

Javed, M., Y. M. Abgaz, et C. Pahl (2013). Ontology change management and identification of change patterns. *Journal on Data Semantics 2*(2-3), 119–143.

Khattak, A. M., K. Latif, et S. Lee (2013). Change management in evolving web ontologies. *Knowledge-Based Systems 37*(0), 1–18.

Klein, M. (2004). *Change Management for Distributed Ontologies*. Ph. D. thesis, Vrije Universiteit Amsterdam, Amsterdam, The Netherlands.

Klein, M. et N. F. Noy (2003). A component-based framework for ontology evolution. In *Workshop on Ontologies and Distributed Systems*.

Liu, L., P. Zhang, R. Fan, R. Zhang, et H. Yang (2014). Modeling ontology evolution with setpi. *Information Sciences 255*, 155–169.

Löwe, M. (1993). Algebraic approach to single-pushout graph transformation. *Theoretical Computer Science 109*(1), 181–224.

Luong, P.-H. et R. Dieng-Kuntz (2007). A rule-based approach for semantic annotation evolution. *Computational Intelligence 23*(3), 320–338.

Mahfoudh, M., G. Forestier, L. Thiry, et M. Hassenforder (2013). Consistent ontologies evolution using graph grammars. In *Knowledge Science, Engineering and Management*, pp. 64–75. Springer.

Mahfoudh, M., G. Forestier, L. Thiry, et M. Hassenforder (2014a). Approche formelle de fusion d'ontologies à l'aide des grammaires de graphes typés. In *Conférence Extraction et Gestion des Connaissances*, pp. 565–568. Hermann-Éditions.

Mahfoudh, M., L. Thiry, G. Forestier, et M. Hassenforder (2014b). Algebraic graph transformations for merging ontologies. In *Model and Data Engineering*, pp. 154–168. Springer.

Rozenberg, G. (1999). *Handbook of graph grammars and computing by graph transformation*, Volume 1. World Scientific.

Stojanovic, L. (2004). *Methods and Tools for Ontology Evolution*. Ph. D. thesis, University of Karlsruhe, Germany.

Summary

Ontology evolution is an essential process in the ontology life cycle. It is defined by different types of ontology changes: basic, composite and complex. The complex and composite changes are very useful in the sense that they help user to adapt his/her ontology without getting lost in the details of the basic changes. However, they hide behind a sophisticated formalization since they affect, at the same time, several ontological entities and may cause inconsistencies to the evolved ontology. To address this issue, this article presents a new formalization of composite and complex ontology changes based on typed graph grammars. This formalization is based on the algebraic approach Single pushout (SPO) of graph transformations and has two main advantages: (1) it provides a new formalization to control the graph transformations and avoid inconsistencies in an a priori manner; (2) it facilitates the description of composite and complex changes while reducing the number of the rewriting rules required to apply them.