Approche d'évolution d'ontologie guidée par des patrons de gestion de changement.

Rim Djedidi

Université Paris-Sud XI
Faculté des Sciences d'Orsay

N° d'ordre : 9651

THESE

présentée pour l'obtention du grade de Docteur en Sciences de
l'Université Paris-Sud XI Orsay

Par

Rim JEDIDI

Spécialité : Informatique

Approche d'évolution d'ontologie guidée par des patrons de gestion de changement

Soutenue le 26 Novembre 2009 devant le jury d'examen :

Mme Marie-Aude AUFAURE, Professeur, Centrale Paris (Directrice de thèse)
Mme Yolaine BOURDA, Professeur, Supélec (Examinateur)
Mme Sylvie DESPRES, Professeur, Université Paris 13 (Rapporteur)
M. Fabien GANDON, Chargé de Recherche – HDR, INRIA Sophia Antipolis (Examinateur)
M. Gilles KASSEL, Professeur, Université de Picardie Jules Verne (Rapporteur)
Mme Chantal REYNAUD, Professeur, Université Paris 11 (Examinateur)

Thèse préparée au sein du
Département Informatique de Supélec,
3 rue Joliot-Curie,
91192 Gif-sur-Yvette Cedex,
FRANCE

Remerciements

Je remercie tout d'abord, Marie-Aude Aufaure, ma directrice de thèse, Professeur à l'école Centrale Paris, pour tous ses conseils, ses recommandations, son soutien et ses encouragements tout au long de cette thèse. Merci de m'avoir guidée dans les différentes étapes de la thèse.

Je remercie Sylvie Desprès, Professeur à l'Université Paris-13, et Gilles Kassel, Professeur à l'Université de Picardie, d'avoir accepté la lourde tâche de rapporter sur ce travail et de faire partie de mon jury. Leur relecture et leurs observations judicieuses m'ont été d'une grande aide.

Je remercie également Chantal Reynaud, Professeur à l'Université Paris-11, et Fabien Gandon, Chargé de Recherche à l'INRIA Sophia Antipolis, d'avoir accepté d'examiner mon travail et de faire partie de mon jury.

Je remercie tout particulièrement Yolaine Bourda, Professeur à l'école Supélec et Directrice du Département Informatique, de m'avoir accueillie dans le département et de m'avoir permis de réaliser cette thèse dans de bonnes conditions, et aussi pour avoir accepté de participer à mon jury de thèse.

Je tiens à remercier tous les membres du Département Informatique et du Centre de Ressources Informatiques de Supélec, et tout particulièrement Cécile, Assia et Luc pour leur amitié et leur soutien constant. Et aussi, Evelyne, Nadjet et Sylvain pour leur aide et leur disponibilité.

Mes remerciements s'adressent également à Cédric Jacquiot pour son amitié et son soutien moral tout au long de cette thèse.

J'exprime ma profonde gratitude à mon cher papa pour son affection et son soutien depuis toujours.

Un grand merci plein de reconnaissance à Jérémie pour son amour, sa tendresse, son soutien, sa compréhension et son encouragement sans limite.

Je remercie finalement toutes les personnes qui m'ont aidé de prés ou de loin, à la réalisation de ce travail.

« *La méthode de la science est une méthode de conjectures audacieuses et de tentatives ingénieuses et sévères pour réfuter celles-ci.* »

Karl Popper, La Connaissance Objective, 1972.

Table des matières

Chapitre 1 : Introduction générale

UN PEU D'HISTOIRE…

La notion d'*Ontologie* est issue de la philosophie du temps d'Aristote. Elle représente une branche de la métaphysique qui se rapporte à l'étude « de l'être en tant qu'être indépendamment de ses déterminations particulières et dépouillé de ses attributs singuliers, et des choses en elles-mêmes, indépendamment de leurs apparences ». [Aristote]

La notion d'ontologie a été réutilisée bien plus tard, dans le domaine de l'intelligence artificielle et plus précisément, dans la branche de l'ingénierie de connaissances, pour s'appliquer à la représentation de connaissances définissant l'ensemble des concepts d'un langage donné, et les relations logiques qu'ils entretiennent entre eux.

De la branche de l'ingénierie de connaissances, a émergé l'ingénierie ontologique comme un nouveau champ de recherche pour développer des systèmes informatiques basés sur des mécanismes manipulant des connaissances et leur sémantique, plutôt que des informations (Mizoguchi & Ikeda, 1997).

Plusieurs définitions ont été proposées dans la littérature pour la notion d'ontologie. Les plus utilisées sont celles de Grüber (Grüber, 1993) définissant une ontologie comme « une spécification explicite d'une conceptualisation » ; complétée par la définition de Borst (Borst, 1997) : « les ontologies sont définies comme une spécification formelle d'une conceptualisation partagée ». Une discussion des différentes définitions proposées dans la littérature est présentée dans (Gomez-Perez et al., 2004).

Ainsi, l'ingénierie ontologique[1] représente l'étude de tout le cycle de vie des ontologies (Davies et al., 2003):

- Le processus de développement d'ontologie ;
- Les méthodologies de construction, d'alignement et d'intégration, de déploiement, d'évaluation et validation, d'exploitation, et de gestion d'évolution d'ontologie, comprenant les méthodes, les techniques, les outils et les langages de représentation (formalisme et expressivité) ;
- L'étude des dépendances entre les ontologies.

Les méthodologies de développement d'ontologie proposées en ingénierie ontologique, ne représentent pas des standards mais plutôt, des approches de construction définissant des principes et des critères, et guidant le développement des ontologies.

Différentes approches ont été proposées dans la littérature (Gomez-Perez et al., 2004). Les premières méthodologies – approches *from scratch* – se basaient sur un processus de construction manuel sans connaissances *a priori* (Uschold & King, 1995) (Güninger & Fox, 1995). Il y a eu ensuite, la méthodologie *METHONTOLOGY*, qui intègre la construction d'ontologie dans un processus complet de gestion de projet, de développement et de support (Fernadez-Lopez et al., 1997). D'autres approches de construction et d'apprentissage d'ontologie se sont basées sur des techniques d'analyse de langages naturels (techniques linguistiques, fouille de textes, techniques hybrides) ou des techniques d'apprentissage machine. Toutes ces méthodologies d'apprentissage d'ontologie, développées dans la littérature, proposent et appliquent des méthodes et des techniques complémentaires qui varient selon les

[1] http://www.aifb.uni-karlsruhe.de/WBS/cte/ontologyengineering/

sources d'entrée (textes, dictionnaires, documents Web, schémas semi-structurés, schémas relationnels, bases de connaissances, instances ou encore des ontologies) et le type des connaissances à extraire (concepts, relations taxonomiques, relations non taxonomiques, axiomes, instances). Le lecteur peut trouver une classification des approches selon le type des connaissances extraites à l'annexe A.

Après cette petite introduction énonçant l'objet sur lequel porte nos travaux de recherche : les ontologies ; nous exposons dans les sections suivantes, les objectifs de départ qui ont motivé la proposition de cette de thèse de doctorat et nous retraçons leur évolution.

I- OBJECTIFS DE DEPART

La construction d'ontologies de domaine étant appréhendée par différentes méthodologies appliquées en fonction de la nature des sources considérées, du niveau d'expressivité visé, de l'utilisation envisagée de l'ontologie à construire, etc. ; l'idée que nous avons eue dans notre projet de recherche de départ était de proposer une approche générique de construction d'ontologies de domaine opérationnelles à partir de sources hétérogènes. Les principaux objectifs de l'approche étaient de fournir une conceptualisation basée sur une large couverture sémantique et une représentation formelle et opérationnelle des connaissances.

La généricité de l'approche était fondée sur l'adaptabilité du modèle de construction d'ontologies proposé à différents domaines d'application, et sur la prise en compte de l'aspect évolutif des ontologies de domaine à travers un processus de construction itératif et incrémental, permettant un enrichissement progressif et continu de la conceptualisation, et tenant compte de l'apparition de nouvelles sources, et de nouveaux besoins de changement.

A partir de ces objectifs, nous avons proposé une architecture d'approche de construction d'ontologies de domaine organisée en trois niveaux : 1) un niveau de *modélisation de domaine* à partir de sources hétérogènes, 2) un niveau de *généralisation et abstraction* et 3) un niveau *d'opérationnalisation et exploitation.*

Le niveau de modélisation de domaine vise à assurer une couverture sémantique large du domaine en tenant compte de sources hétérogènes et d'organiser et structurer l'ensemble des concepts et des relations issus de ces différentes sources. Ces sources hétérogènes regroupent des données structurées notamment des ontologies existantes, des données semi-structurées et des sources de données textuelles informelles. L'idée est de commencer par la capitalisation des connaissances formelles existantes en réutilisant des ontologies existantes, et d'enrichir et compléter par la suite l'ontologie de domaine obtenue, en intégrant des connaissances extraites des autres sources considérées.

Le niveau de généralisation et abstraction permet de définir – si besoin – les concepts les plus abstraits du domaine (niveaux supérieurs proches de la racine de l'ontologie), et de généraliser la conceptualisation obtenue au niveau précédent, en ayant recours à une ontologie noyau du domaine (Domain Core-Ontology) offrant une représentation générique établie à un haut niveau d'abstraction. A ce niveau, l'objectif est de regrouper des classifications, préciser leur origine (racine), et de faciliter ainsi une éventuelle réutilisation de l'ontologie de domaine construite.

Le niveau d'opérationnalisation et exploitation tient compte des dépendances aux objectifs opérationnels. Il permet – si besoin – d'enrichir et d'ajuster la sémantique de l'ontologie construite aux besoins liés à son usage dans un contexte applicatif bien déterminé. Ainsi, à partir de la conceptualisation obtenue (au niveau 1 ou aux niveaux 1,2), une ontologie référentielle du domaine intégrée dans son environnement d'application pour une utilisation réelle, est définie à ce niveau. Une ontologie référentielle est une ontologie de domaine opérationnalisée permettant de représenter les contraintes qui spécifient l'interprétation de ses connaissances, et de l'adapter au contexte de son application. En effet, le contexte d'usage peut influencer certains choix ontologiques. Les besoins liés à un système d'indexation de documents par exemple, sont différents de ceux liés à un système d'aide à la décision. Pour le premier,

nous avons besoin d'une ontologie de granularité fine spécialisant les classifications afin d'assurer une meilleure indexation. Pour le second type de système, nous avons besoin d'une conceptualisation riche en relations sémantiques et en expressivité axiomatique afin de guider la prise de décision tel qu'inférer des relations entre symptômes et traitements dans une application médicale.

L'architecture est illustrée par la figure ci-après (figure 1).

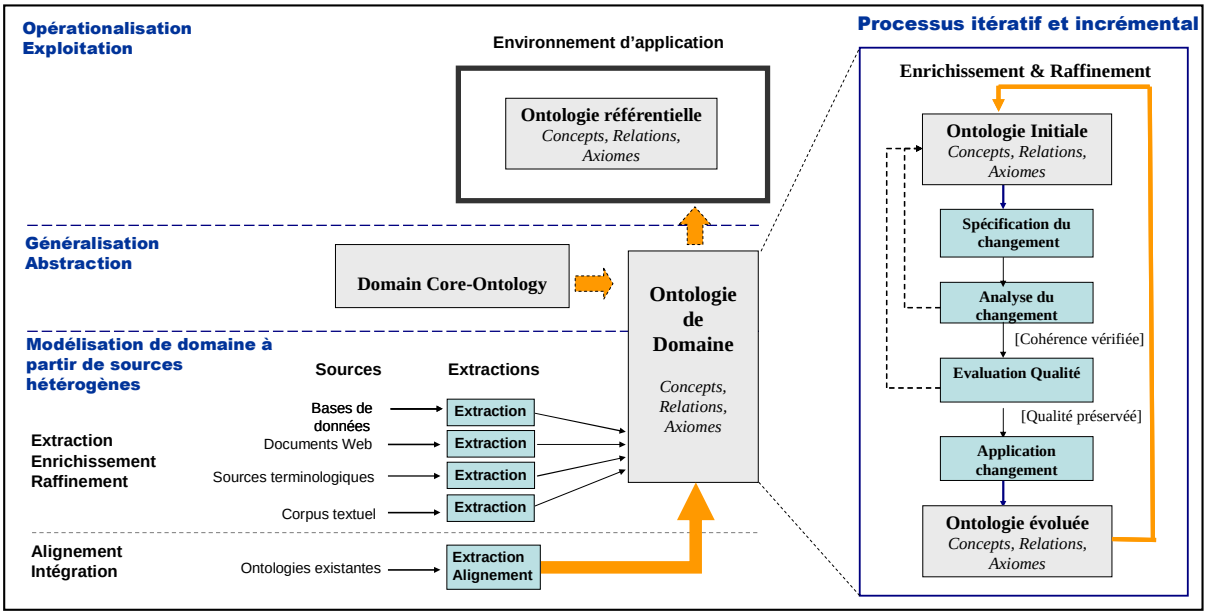


Figure 1. Proposition d'une approche générique de construction d'ontologie à partir de sources hétérogènes.

II- EVOLUTION DES OBJECTIFS

En se focalisant sur les besoins en enrichissement et raffinement d'ontologie, nous avons défini une première ébauche d'un processus itératif et incrémental intégrant des étapes de spécification de changement, de vérification de la cohérence et d'évaluation de l'impact de changement sur la qualité de l'ontologie afin de ne valider que les changements cohérents et préservant ou améliorant la qualité de l'ontologie enrichie. Les problématiques inhérentes à la gestion des changements d'ontologies et les solutions potentielles qui peuvent être apportées, ont suscité notre motivation à recentrer les objectifs de la thèse et les orienter sur l'évolution d'ontologie de domaine.

III- PROBLEMATIQUE POSEE

Appliquer des changements à une ontologie vise normalement, à la modifier pour la rendre plus précise et plus adéquate au domaine qu'elle modélise et à ses objectifs d'usage. La modification d'ontologie présente plusieurs difficultés aussi bien sur un plan pratique que théorique. Il n'est pas toujours évident d'appréhender clairement ce qui est attendu d'un besoin de changement, ni comment un changement peut être réalisé. Le coût de l'adaptation des ontologies aux changements appuie la nécessité de gérer l'évolution d'ontologie de manière méthodique et formelle.

Nous nous intéressons aux problèmes inhérents à la gestion des changements d'une ontologie de domaine dans un contexte local (sans tenir compte d'une dimension collaborative ou distribuée de

l'environnement de l'ontologie). Conduire l'application des changements tout en maintenant la cohérence de l'ontologie est une tâche cruciale et couteuse en termes de temps et de complexité. Un processus automatisé est donc essentiel. Mais comment conduire l'application d'un changement tout en assurant la cohérence de l'ontologie évoluée ? Comment analyser les effets d'un changement et les résoudre ? Et si plusieurs solutions sont possibles, laquelle choisir et selon quels critères ?

IV- APPROCHE PROPOSEE

Pour répondre à toutes ces questions, nous avons défini une méthodologie de gestion de changement *Onto-Evo*^{a}*l* (Ontology Evolution-Evaluation) qui s'appuie sur une modélisation à l'aide de patrons. Ces patrons spécifient des classes de changements, des classes d'incohérences et des classes d'alternatives de résolution. Sur la base de ces patrons et des liens entre eux, nous proposons un processus automatisé permettant de conduire l'application des changements tout en maintenant la cohérence de l'ontologie évoluée. La méthodologie intègre aussi une activité d'évaluation basée sur un modèle de qualité d'ontologie. Ce modèle est employé pour guider la gestion des incohérences en évaluant l'impact des résolutions proposées sur la qualité de l'ontologie et ainsi choisir celle qui préserve la qualité de l'ontologie évoluée.

La gestion des changements étant fortement liée au modèle dans lequel est représentée l'ontologie, nous nous focalisons sur le langage OWL et nous tenons compte de l'impact des changements sur la cohérence logique de l'ontologie telle que spécifiée dans la couche OWL DL.

V- PLAN DE LA THESE

Le contenu de la thèse est organisé en huit chapitres : un chapitre Introduction générale, un chapitre Conclusion et perspectives et six chapitres développant l'état de l'art et les différents travaux menés.

Le chapitre 2 : Etat de l'art de l'évolution d'ontologie met l'accent, à travers une vue globale des recherches existantes, sur les problématiques inhérentes aux changements d'ontologie et détaille un état de l'art des approches d'évolution d'ontologie classées selon les questions qu'elles abordent et le modèle ontologique qu'elles considèrent (langage de représentation d'ontologie). Le chapitre présente également, les outils de développement d'ontologie intégrant des fonctionnalités d'évolution et de maintenance.

Le chapitre 3 : *Onto-Evo*^{a}*l* : une approche d'évolution d'ontologie présente l'approche d'évolution d'ontologies définie en énonçant ses principes, décrivant son architecture globale, et en détaillant le processus de gestion de changement et ses différentes phases, ainsi que la modélisation de l'historique d'évolution.

Le chapitre 4 : CMP : Patrons de gestion de changement d'ontologie détaille la modélisation des patrons de gestion de changement CMP (Change Management Patterns) et les liens conceptuels entre eux. Ces patrons correspondent à des types de changements, d'incohérences et d'alternatives de résolution gérés par l'approche *Onto-Evo*^{a}*l* et spécifiés formellement par l'ontologie CMP. Ils sont employés dans le processus de gestion de changement comme des méta-connaissances offertes pour guider et contrôler l'application des changements tout en assurant la cohérence logique de l'ontologie évoluée.
Les patrons CMP sont représentés selon deux couches : une couche de présentation sous forme d'un catalogue de patrons décrits en langage naturel et illustrés par des diagrammes UML, et une couche de spécification formelle sous forme d'une ontologie OWL DL définissant la sémantique des patrons, des relations entre eux, et de leur application.

Le chapitre 5 : Résolution des incohérences détaille la résolution des incohérences dans l'approche *Onto-Evo*^{a}*l* en expliquant comment sont sélectionnés, instanciés et employés les patrons CMP pour la gestion d'un changement, et en décrivant le procédé de résolution des incohérences et de proposition de résolutions. Une synthèse des travaux existants en débogage d'ontologie y est aussi présentée.

Le chapitre 6 : Evaluation des résolutions détaille l'ensemble des caractéristiques, critères et métriques définissant le modèle de qualité sur lequel se base l'évaluation des résolutions dans l'approche *Onto-Evo*^{a}*l*, et décrit l'emploi de l'évaluation de qualité pour le guidage de la résolution d'un changement. Une synthèse des travaux existants en évaluation d'ontologie y est aussi présentée.

Le chapitre 7 : Expérimentation et implémentation décrit l'évaluation de l'approche *Onto-Evo*^{a}*l* à travers une première évaluation manuelle basée sur des ontologies test et une deuxième évaluation automatique employant le prototype implémenté. Un modèle pour une architecture complète du système d'évolution d'ontologie est proposé, de même qu'un contexte d'application du système comme plugin d'une plateforme de construction et d'édition d'ontologie est présenté.

Chapitre 2 : Etat de l'art de l'évolution d'ontologie

Résumé : Tout au long de leur cycle de vie, les ontologies évoluent pour répondre à différents besoins de changement. Plusieurs problèmes émanent de l'évolution d'ontologie : l'identification des besoins de changement, l'explicitation des changements, l'analyse et la résolution des impacts de changement, la validation des changements, la traçabilité des changements, la propagation des changements aux artefacts dépendants, la gestion des versions d'ontologie, etc. A travers une vue globale des recherches en évolution d'ontologie, l'objectif de ce chapitre est de mettre l'accent sur les problématiques inhérentes aux changements d'ontologie ; de détailler un état de l'art des approches d'évolution d'ontologie classées selon les questions qu'elles abordent et le modèle ontologique qu'elles considèrent (langage de représentation d'ontologie) ; et de présenter les outils de développement d'ontologie intégrant des fonctionnalités d'évolution et de maintenance.

I- INTRODUCTION

La portée des ontologies est aujourd'hui de plus en plus large. Outre le Web sémantique, elles sont aussi appliquées pour des systèmes de gestion de connaissances, de gestion de documents et de contenu, d'intégration d'informations et de modèles, etc. Elles offrent à ces systèmes une conceptualisation sémantique riche et explicite, des possibilités de raisonnements et d'exploitation de requêtes et facilitent l'interopérabilité entre systèmes. Les connaissances ontologiques ne peuvent cependant pas être considérées comme statiques et finies. Elles ont besoin d'être mise à jour continuellement. Au cours de leur développement, les ontologies sont construites selon un processus dynamique donnant lieu à une première ontologie minimale qui est par la suite révisée, raffinée et complétée (Noy & McGuinness, 2001). De même en cours d'usage, l'ontologie évolue pour prendre en compte les évolutions du domaine qu'elle modélise ainsi que les changements de son environnement d'application.

L'objectif de ce chapitre est de présenter une analyse des travaux existants en évolution d'ontologie. Le chapitre est composé de trois parties principales. Dans la première partie (section II), nous exposons les besoins en évolution d'ontologie, nous présentons une étude comparative de l'évolution des ontologies, des schémas de bases de données et des systèmes de bases de connaissances et nous détaillons les problématiques de gestion de changements d'ontologie. Dans la deuxième partie, nous présentons une étude des approches existantes en évolution d'ontologie en analysant les fonctionnalités supportées par leur processus (sections III). L'étude prend aussi en compte le langage d'ontologie considéré par ces approches et ses contraintes de cohérences. Nous décrivons également les principaux outils de développement d'ontologie supportant une partie ou la totalité du processus d'évolution d'ontologie (sections IV). Dans la troisième partie, nous synthétisons par un tableau comparatif, l'ensemble des approches existantes et les outils qui les implémentent (sections V) et nous discutons les principales observations à retenir et questions ouvertes à approfondir (sections IV).

II- PROBLEMATIQUES D'EVOLUTION D'ONTOLOGIE

La conceptualisation modélisée par une ontologie renseigne les applications qui en dépendent, sur la façon d'utiliser les données liées aux connaissances de l'ontologie. Mais, ni les données, ni l'ontologie elle-même ne sont stables. Les changements d'ontologie ont un effet direct sur la manière d'interpréter les données. L'évolution des données doit être propagée à l'ontologie afin d'assurer la pertinence de celle-ci par rapport aux applications pour lesquelles elle a été construite.

Dans cette section, nous tentons de susciter une meilleure compréhension de l'évolution d'ontologie en soulevant ses différentes problématiques, en la confrontant aux problématiques et solutions identifiées dans des domaines connexes et en analysant ses caractéristiques.

L'évolution d'ontologie engage une gestion des changements d'ontologie tout en maintenant la cohérence. Elle est définie comme étant « l'adaptation – dans le temps – d'une ontologie aux besoins de changement et la propagation cohérente des changements aux artefacts dépendants » (Maedche et al., 2003, p. 287).

II.1- Besoins en évolution d'ontologie

Les besoins en évolution d'ontologie ont été abordés par plusieurs travaux (Blundell & Pettifer, 2004) (Noy & Klein, 2004) (Stojanovic & Motik, 2002) (Stojanovic, 2004) (Klein, 2004). L'évolution d'ontologie est une problématique complexe (figure 1). Outre l'identification même des besoins de changement à partir de différentes sources (domaine, environnement d'usage, conceptualisation interne, etc.), la gestion de l'application d'un changement – de sa formulation jusqu'à son application et sa validation finale – nécessite de spécifier le changement requis, d'analyser ses effets sur la cohérence de l'ontologie et les résoudre, de l'implémenter et de valider son application finale. Dans un contexte collaboratif ou distribué, il est aussi nécessaire de propager le changement appliqué localement à l'ontologie, aux artefacts dépendants (les objets, applications et ontologies dépendants) et de valider les changements globalement. De plus, la traçabilité des changements doit être gardée pour pouvoir les justifier, les expliquer, voire les annuler et gérer les différentes versions d'une ontologie.

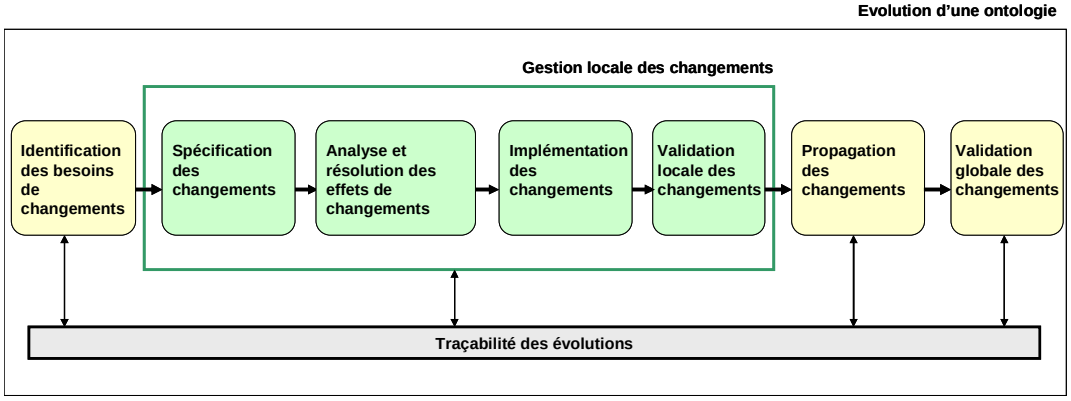


Figure 1. Besoins en évolution d'ontologie.

Avant d'approfondir les problématiques en gestion de changements d'ontologie, nous nous proposons de comparer dans ce qui suit, l'évolution d'ontologie aux stratégies existantes en évolution de schéma de bases de données et de système de bases de connaissances.

II.2- Comparaison à l'évolution de schéma de bases de données et de système de bases de connaissances

La gestion de changements a fait l'objet de plusieurs travaux en évolution de schéma de bases de données (Roddick, 1996) (Breche & Wörner, 1995) (Banerjee et al., 1987) et en maintenance de système de bases de connaissances (Menzis, 1999). Plusieurs problématiques de ces domaines sont aussi pertinentes dans le contexte d'évolution d'ontologie. Dans les sous-sections suivantes, nous synthétisons deux études comparatives abordant l'évolution des ontologies vs. respectivement celle des schémas de bases de données et celle des systèmes de bases de connaissances.

II.2.1- Evolution de schéma de bases de données vs. évolution d'ontologie

L'évolution d'ontologie est un domaine très proche de l'évolution de schémas de bases de données (Roddick, 1996) particulièrement les bases de données orientées objets (Banerjee et al., 1987) (Franconi et al., 2000). Les idées principales en modélisation de version de bases de données sont aussi pertinentes pour les ontologies. Cependant, les problématiques de changement d'ontologie sont de plus grande ampleur et nécessitent de prendre en compte leurs particularités. Une étude comparative de l'évolution des schémas de bases de données vs. celle des ontologies est synthétisée dans le tableau suivant (Table 1) :

Schémas de bases de données (BD)	Ontologies
– Opérations de changement assez fréquentes durant le cycle de vie du système de BD	– La fréquence d'évolution peut être très conséquente notamment si la découverte des changements est guidée par les utilisateurs.
– La gestion de l'évolution n'est pas assurée formellement, c'est l'administrateur de BD qui doit veiller sur la maintenance de la cohérence.	– Guidage de l'ingénieur d'ontologie pour la détection et l'abstraction des changements.
– Adaptation manuelle des instances.	– Automatisation de la résolution des changements et notification à l'ingénieur d'ontologie des effets du changement.
L'évolution d'ontologie s'apparente plus à l'évolution de schéma de BD orientées objets que celle de schéma de BD relationnelles, les BD orientées objets étant sémantiquement plus riches à travers les principes d'héritage et de hiérarchie.	
– Les schémas orientés objet reflètent la structure d'un ensemble de données et de codes et tiennent compte du comportement des objets en intégrant les méthodes dans le modèle. De plus, la représentation physique des données y est intégrée (entier réel, etc.).	– Les ontologies reflètent la structure d'un domaine à travers des concepts, des propriétés et des contraintes sur ces primitives.
Plusieurs travaux en évolution d'ontologie ont adopté des approches existantes en schémas de BD orientées objets. Cependant, les différences entre ces deux modèles font que les approches d'évolution d'ontologie soient considérées comme une extension plutôt qu'une adaptation des approches existantes (Noy & Klein, 2004).	
– Les instances (objets BD) ne sont pas au même niveau que les classes.	– Le niveau terminologique (classes et propriétés) et le niveau assertionnel (instances) ne sont pas délimités. Les instances et les classes peuvent être traitées et utilisées ensemble comme pour le traitement des requêtes basées sur les ontologies.

	L'effet des changements sur les requêtes est donc à prendre en compte (Klein, 2004).
– L'intégration d'un schéma dans un autre n'est pas possible.	– La réutilisation d'ontologie permet d'intégrer des ontologies ou parties d'ontologie dans d'autres ontologies. Ainsi, les changements permettent d'inclure/exclure une ontologie dans/de une autre (Stojanovic, 2004).
– Les changements sont définis dans le modèle lui même.	– Outre la représentation des changements dans le modèle de l'ontologie, il est essentiel de spécifier la sémantique des changements à travers les conditions à vérifier et les actions à appliquer pour maintenir la cohérence (Stojanovic, 2004). La structure des ontologies fait que les changements impliquent plus d'incohérences et que les résolutions soient plus complexes.
– L'évolution nécessite la définition d'un modèle de cohérence et la spécification explicite des changements à appliquer.	– Les modèles d'ontologie étant plus riches, la définition de la cohérence d'ontologie implique plus de contraintes de même que les possibilités de changements sont plus importantes qu'en modèles relationnels ou orientés objets (Stojanovic, 2004) : dans une ontologie, une propriété par exemple, peut être définie sans être attachée à un concept.
– La sémantique des schémas n'est pas assez explicite pour faciliter l'application des mécanismes de raisonnement vérifiant la cohérence.	– La sémantique des ontologies est plus explicite et permet l'application de mécanismes de raisonnement pour la détection d'incohérences.
Les solutions proposées pour le contrôle des effets de changements en évolution de schéma de BD sont basées sur des règles à satisfaire pour maintenir la cohérence et sur des mécanismes d'inférences basés sur des axiomes (Klein, 2004). Les approches de résolution des changements d'ontologie combinent (en fonction du modèle d'ontologie) ces deux solutions complémentaires.	
– Pour propager les changements aux instances, trois solutions sont proposées (Stojanovic, 2004): (i) la migration des données et l'adaptation immédiate des instances au schéma ; (ii) des mécanismes de synchronisation de schéma – instances (conversion différée, propagation des changements à travers des objets de conversions différées, gestion de versions en cas de non propagation de changements et si tout objet est affecté aux différentes versions) et (iii) la combinaison des deux solutions précédentes.	– Pour la propagation des changements d'ontologie, deux scenarii sont distingués (Stojanovic, 2004): (i) un scénario centralisé basé sur la migration des données et (ii) un scénario distribué basé sur la conversion différée. Le choix entre les deux scenarii dépend de la priorité accordé aux deux critères contradictoires : cohérence global et temps d'exécution.
– La propagation des changements est limitée aux instances.	– La propagation des changements s'étend à tous les artefacts dépendant de l'ontologie évoluée c'est-dire les instances, les annotations, les applications et les ontologies dépendantes.

Table 1. Evolution de schéma de BD vs. évolution d'ontologie.

II.2.2- Evolution de système de bases de connaissances vs. évolution d'ontologie

Pour étudier les problématiques d'évolution d'ontologie, il est aussi intéressant de considérer les approches existantes de gestion des changements de structures formelles de connaissances (Coenen & Bench-Capon, 1993) (Menzis, 1999) et de comparer l'évolution de ces structures à celle des ontologies. Les systèmes à base de connaissances succédant aux systèmes experts, répondent à un besoin de stockage et consultation de connaissances, de raisonnement sur ces connaissances, de modification de ces connaissances et de partage de connaissances entre systèmes. La maintenance d'un système à base de connaissances (SBC) se base sur la maintenance de sa base de connaissances elle-même (Menzis, 1999). Une étude comparative de l'évolution des SBC vs. celle des ontologies est synthétisée dans le tableau suivant (Table 2) :

Systèmes de bases de connaissances (SBC)	**Ontologies**
– La principale caractéristique des SBC est la séparation modulaire des connaissances de domaine et des connaissances de raisonnement décrivant les règles heuristiques d'utilisation des connaissances de domaine (Charlet et al., 2004). Les connaissances de raisonnement représentent des méthodes d'inférence fournies par les raisonneurs et exprimées généralement en code procédural.	– Les connaissances de domaine d'un SBC peuvent être représentées par une ontologie (Charlet et al., 2004), et les approches de maintenance des SBC sont adaptables à l'évolution des ontologies (Stojanovic, 2004). Cependant, outre la richesse du modèle ontologique, la réutilisation et l'intégration d'ontologies existantes dont certaines distribuées sur le Web rendent l'évolution d'ontologie beaucoup plus complexe.
La maintenance peut être classée en quatre groupes (IEEE, 1990) (Stojanovic, 2004): une maintenance adaptative, une maintenance perfective, une maintenance corrective et une maintenance préventive.	
– La maintenance adaptative résulte des changements de l'environnement du SBC ou d'une meilleure compréhension du domaine de connaissances.	– La maintenance adaptive permet d'adapter l'ontologie aux changements du domaine et d'acquérir de nouvelles connaissances.
– La maintenance perfective a pour objectif de répondre aux besoins des utilisateurs.	– L'identification des besoins de changement d'ontologie peut être guidée par les besoins des utilisateurs (Völker et al., 2005) (Völker et al. 2007a) (Völker et al., 2007b).
– La maintenance corrective permet de répondre à un comportement inapproprié du SBC et de résoudre des erreurs d'ordre syntaxiques, sémantiques ou d'identification des connaissances stockées dans la base.	– La maintenance corrective permet d'assurer la cohérence et prend en compte l'usage de l'ontologie (Stojanovic, 2004).
– La maintenance préventive permet d'éviter des problèmes futurs en analysant la structure de la base de connaissances afin de révéler d'éventuelles erreurs.	– La maintenance préventive a été adaptée au raffinement d'ontologie. L'identification des besoins de changement est guidée par la structure et l'usage de l'ontologie (Stojanovic, 2004).

Table 2. Evolution de système de bases de connaissances vs. évolution d'ontologie.

II.3- Changements d'ontologie

Dans cette section, nous analysons les changements d'ontologie en présentant les activités de changement d'ontologies, les effets de changement et les différentes classifications des changements.

II.3.1- Activités de changement

La gestion des changements d'ontologie peut être considérée comme un éventail d'activités de changement. La distinction entre ces différentes activités est relativement ambigüe. En considérant le rôle de l'évolution dans le cycle de vie d'une ontologie et en adoptant la terminologie de la communauté bases de données (Roddick, 1996), ces activités pourraient être organisées séquentiellement pour produire la ligne de vie d'une ontologie (O'Brien & Abidi, 2006). Quatre principales activités ont été distinguées dans la littérature : révision d'ontologie, gestion de versions d'ontologie appelée aussi *versioning*, adaptation d'ontologie et évolution d'ontologie.

II.3.1.1- Révision d'ontologie

La révision d'ontologie vise à changer l'état d'une ontologie afin d'adapter ses caractéristiques de représentation au domaine modélisé (Klein & Fensel, 2001). Elle correspond à des manipulations logiques qui étendent le niveau d'abstraction d'une ontologie comme par exemple, ajouter ou supprimer des concepts ou encore refactoriser des propriétés (Foo, 1995).

II.3.1.2- Versioning ou gestion de versions d'ontologie

Le versioning consiste à créer et gérer différentes versions d'une ontologie en traitant les incompatibilités entre les différentes versions, les instances et les applications et ontologies qui en dépendent.

L'existence (ou la génération) de plusieurs versions d'une ontologie peut être considérée dans deux principales situations : (i) quand une ontologie distribuée est développée de manière indépendante, plusieurs versions peuvent en résulter (Klein & Fensel, 2001) et (ii) l'ontologie évoluée, résultant de l'application d'un changement, peut être considérée comme une nouvelle conceptualisation du domaine et donc comme une nouvelle version. Néanmoins, la décision revient à l'ingénieur d'ontologie.

Maintenir plusieurs versions d'une ontologie nécessite de disposer de moyens pour différentier les différentes versions et assurer la validité des instances. L'idéal serait de préserver les différentes versions d'une ontologie et de garder toutes les informations concernant les différences et les compatibilités entre elles. Ceci nécessite des méthodes d'identification de versions, de différentiation de versions (basées sur les mêmes principes que les méthodes de mesure de similarités sémantiques en alignement d'ontologie), et de spécification de relations entre versions ; des procédures de mise à jour d'ontologie ; et des mécanismes d'accès aux différentes versions d'une ontologie (Klein & Noy, 2003).

II.3.1.3- Adaptation d'ontologie

L'adaptation d'ontologie a pour objectif de faciliter la gestion des changements fréquents des représentations de concepts, et d'automatiser le processus de production d'adaptateurs de « versions d'intérêt » (O'Brien & Abidi, 2006). C'est l'adaptation locale d'une ontologie aux spécificités d'usage ou à un sous-ensemble des instances de son environnement, en évitant toute contradiction avec la version précédente.

L'adaptation peut être considérée comme une révision pratique selon laquelle une ancienne version est d'abord adoptée puis adaptée à une application spécifique (O'Brien & Abidi, 2006). Les changements d'adaptation sont, pour la plupart, générés par des approches d'identification de changements dirigée par l'utilisateur (voir section III.1.2).

II.3.1.4- Evolution d'ontologie

L'évolution d'ontologie consiste à prendre en compte des changements « persistants » pour faire face à de nouveaux besoins, et à produire de nouvelles versions. La modification de l'ontologie est gérée en maintenant la cohérence de l'ontologie ; en assurant la traçabilité des changements et en sauvegardant leur historique pour fournir des correspondances (mappings) entre versions (Stojanovic et al., 2002a) ; et en contrôlant l'utilisation des instances (Stojanovic et al., 2002b).

II.3.2- Effets de changement

Changer le comportement d'un concept ou d'une relation de sorte que de nouvelles connaissances prennent effet, fait intervenir des connaissances inter-reliées et nécessite la définition de mécanismes spécifiant comment les connaissances peuvent être changées et comment maintenir leur cohérence. Toutefois, les effets de changements doivent être analysés non seulement par rapport à l'ontologie elle-même, mais aussi en tenant compte des raisons de changement et des objectifs d'usage de l'ontologie.

Les effets de changements ont été considérés dans la littérature selon différentes perspectives. Selon une optique d'entrepôts de données basés sur des ontologies, deux catégories d'effets de changements ont été distinguées (Xuan et al., 2006) :

- L'évolution normale selon laquelle l'enrichissement d'une ontologie par de nouvelles connaissances, ne remet pas en cause les connaissances existantes. C'est le principe de la « continuité ontologique » ;
- La révolution lorsque les changements peuvent infirmer des axiomes.

Selon une approche de gestion d'ontologies distribuées (Klein, 04), les effets de changements dépendent de ce que nous avons besoin de préserver dans l'ontologie :

- Les données, pour maintenir les instances entre les différentes versions d'une ontologie ;
- L'ontologie elle-même, pour maintenir les résultats de requêtes de sorte que le résultat d'une requête q_1 sur une version O_i de l'ontologie soit inclus dans le résultat de la même requête q_1 sur une version O_{i+1} de la-dite ontologie ;
- Les conséquences, pour maintenir les inférences de sorte que les faits inférés d'un axiome a_1 sur une version O_i de l'ontologie soient aussi inférés du même axiome a_1 sur une version O_{i+1} de la dite ontologie ;
- La cohérence, pour assurer que la nouvelle version ne contienne pas d'incohérences logiques.

En comparaison avec les changements de schémas de bases de données, les effets de changements peuvent aussi être classés en tenant compte des instances (Noy & Klein, 2004). Il en ressort des :

- Changements préservants lorsque les instances ne sont pas perdues (par exemple en ajoutant un concept ou une propriété) ;
- Changements translatants qui préservent les instances en translatant les connaissances sous une autre forme (en groupant par exemple, des concepts en une union de leurs super-concepts, sous-concepts et propriétés; les instances peuvent être préservées);
- Changements non-préservants lorsque les instances sont perdues (la suppression d'une propriété par exemple, peut causer la perte des instances).

La gestion des effets de changement sous-entend non seulement la vérification de la cohérence mais aussi sa maintenance. La maintenance de la cohérence consiste à proposer – et appliquer – un ensemble de changements additionnels pour résoudre les incohérences. Elle est généralement assurée manuellement par l'ingénieur d'ontologie à travers l'éditeur d'ontologie (Stojanovic & Motik, 2002) (Sure, 2002).

II.3.3- Classification des Changements

L'objectif de la classification des changements est de définir, pour un langage de représentation d'ontologie donné, une taxonomie de changements spécifiant des classes de changements et leurs propriétés. Les principales classifications définies dans la littérature ont été proposées pour les langages KAON[2] (Maedche et al., 2002; Stojanovic, 2004) et OWL[3] (Klein, 2004).

L'ontologie des changements KAON classifie les changements selon trois niveaux d'abstraction (Stojanovic, 2004):

- Changements élémentaires appliquant des modifications à une seule entité de l'ontologie;
- Changements composites appliquant des modifications au voisinage direct d'une entité de l'ontologie ;
- Changements complexes appliquant des modifications à un ensemble arbitraire d'entités de l'ontologie.

Par ailleurs, les changements KAON ont été aussi classés selon leur effet (Stojanovic, 2004): des *changements additifs* ajoutant de nouvelles entités à l'ontologie sans altérer les entités existantes, et des *changements soustractifs* supprimant certaines entités ou parties d'entités. Pensée comme un ensemble minimal et complet, l'ontologie de changements KAON ne tient pas compte des modifications d'entités. Ce type de changements est interprété comme un changement « *renommer* » altérant uniquement les informations lexicales relatives à une entité ou un changement d'« *ensemble* » dépendant de l'entité à modifier.

Une taxonomie similaire pour les ontologies OWL a été présentée dans (Klein, 2004). Cependant, l'ontologie des changements OWL comprend des opérations de changement de type *modifier* ainsi que *mettre à* (Set) et *annuler* (Unset) pour les caractéristiques de propriétés (telles que la symétrie). Klein (2004) différencie des opérations de changements basiques et de changements complexes :

- Les changements basiques sont des changements simples et atomiques qui peuvent être spécifiés en se basant uniquement sur la structure de l'ontologie et qui ne modifient qu'une seule caractéristique du modèle de connaissance OWL, c'est-à-dire une seule entité de l'ontologie (comme une opération d'ajout d'une classe ou de suppression d'une relation « *is-a* ».) ;
- Les changements complexes correspondent à des changements composites et riches groupant des séquences logiques de changements basiques et incorporant des informations sur leur impact sur le modèle logique de l'ontologie (comme par exemple remonter des sous-classes, élargir le co-domaine d'une propriété objet à ses superclasses, fusionner des classes, etc.). Outre dans leur spécification, la complexité se manifeste aussi dans les effets de ces changements. Si les effets des changements basiques restent relativement mineurs, les effets cumulés de tous les changements intermédiaires réalisant un changement complexe sont conséquents.

L'ontologie des changements OWL définit tous les changements basiques possibles étant dérivés exhaustivement du méta-modèle de OWL. Par contre, elle ne définit qu'un sous-ensemble des changements complexes, ceux-ci étant infinis puisque de nouvelles compositions de changements peuvent toujours être définies.

[2] KArlsruhe ONtology: infrastructure open-source de gestion d'ontologie pour des applications business: Kaon.semanticweb.org

[3] http://www.w3.org/TR/owl-features/

III- APPROCHES D'EVOLUTION D'ONTOLOGIE

Divers travaux ont discuté des caractéristiques d'un processus d'évolution d'ontologie (Stojanovic et al., 2002a) (Stojanovic et al., 2002b) (Klein, 2004) et plusieurs approches ont été proposées. Certaines de ces approches traitent des problématiques particulières de gestion de changement telles que l'identification des besoins de changement (Stojanovic et al., 2003a) (Cimiano & Völker, 2005) (Bloehdorn, et al., 2006), la détection des changements et la sauvegarde des versions (Klein et al., 2002a) (Noy et al., 2004) (Plessers & De Troyer, 2005) (Eder & Wiggisser, 2007), la spécification formelle des changements (Stojanovic et al., 2002c) (Klein, 2004) (Plessers et al., 2007), l'implémentation des changements (Stojanovic et al., 2003b) (Stojanovic, 2004) (Flouris, 2006), la maintenance de la cohérence (Stojanovic, 2004) (Haase & Stojanovic, 2005) (Haase & Völker, 2005) (Plesser & De Troyer, 2006), le versioning d'ontologie (Klein & Fensel, 2001) (Klein et al., 2002a) (Klein, 2004). D'autres, proposent un processus – plus ou moins – global d'évolution comprenant aussi bien l'analyse et la résolution des impacts de changement que la propagation des changements aux artefacts dépendants de l'ontologie évoluée c'est-à-dire les objets, les ontologies et les applications qu'elle référence (Stojanovic, 2004) (Klein, 2004) (Bloehdorn et al., 2006) (Luong, 2007).

Dans cette section nous présentons les principales de ces approches existantes et nous analysons les fonctionnalités qu'elles supportent par leur processus.

III.1- Approche d'apprentissage d'ontologie basée sur l'identification des besoins de changement

Les ontologies sont de plus en plus appliquées à des environnements ouverts et dynamiques, les connaissances qu'elles modélisent ainsi que les besoins utilisateurs auxquels elles répondent sont en continuelle évolution. Pour répondre à leurs objectifs organisationnels ainsi qu'aux attentes des usagers, les ontologies doivent s'adapter aux changements. L'exploration des sources de changements et l'identification des besoins de changement sont en effet, un axe de recherche primordial.

Un besoin de changement peut émaner de la dynamicité de l'environnement applicatif de l'ontologie, de l'évolution du domaine qu'elle modélise, de la modification des besoins utilisateurs, de la prise en compte d'une connaissance nouvelle ou précédemment non disponible, des corrections et restructurations apportées à la conceptualisation de l'ontologie, ou encore de la réutilisation de l'ontologie pour d'autres applications. Dans un contexte distribué, l'évolution d'une ontologie – pour l'une de ces raisons de changements – pourrait nécessiter la modification des ontologies dépendantes afin de rendre compte des potentiels changements de terminologie ou de représentation (Heflin et al., 1999).

Notons que la détection des changements et l'identification des besoins de changement sont deux approches distinctes. La première, vise à découvrir les changements d'ontologie – déjà appliqués – pour des objectifs de versioning par exemple. La deuxième approche a pour objectif de générer des changements d'ontologie à partir de besoins explicites ou implicites de changement. Les besoins explicites peuvent être définis par l'ingénieur d'ontologie pour adapter l'ontologie à de nouveaux besoins ou par les utilisateurs de l'ontologie en se basant sur des feedbacks d'usage. Les changements résultant de besoins explicites sont appelés des changements *descendants* (*top-down*) (Bloehdorn et al., 2006). Les besoins implicites sont déduits de l'analyse du comportement du système et sont appelés des changements *ascendants* (*top-down*) (Bloehdorn et al., 2006).

L'approche présentée dans cette section, aborde l'évolution d'ontologie à travers la dynamicité d'un processus d'apprentissage d'ontologie et met l'accent sur la génération des changements (Cimiano, 2007). Le processus d'apprentissage est appliqué comme un affinement d'ontologie. Il tient compte de

l'évolution des données et des connaissances. L'identification des besoins de changement est dirigée par les données (*data-driven*) en se basant sur les corpus ou par les utilisateurs (*user-driven*). La traçabilité des changements (quels changements, par qui, pourquoi, etc.) est aussi prise en compte afin de garder les explications et justifications des changements et même des références aux segments du corpus ayant déclenché les changements.

III.1.1- Identification de besoins de changement dirigée par les données

L'identification des besoins de changement dirigée par les données (*data-driven*) consiste à dériver des changements d'ontologie à partir des modifications observées dans les sources de connaissances de l'ontologie (Bloehdorn et al., 2006).

L'identification des besoins de changement dirigée par les données peut être organisée selon quatre phases (Cimiano & Völker, 2005):

1. Une phase de traitement du corpus pour formuler un besoin de changement : plusieurs algorithmes sont proposés (extraction de termes, apprentissage de taxonomies et de relations, etc.). L'exécution de ces algorithmes est guidée par un *entrepôt de références de changement* (change reference store) et des *preuves justifiant les changements* (change evidence) ;
2. Une phase de stratégies de changement étudiant comment les changements dans le corpus affectent les différentes entités de l'ontologie ;
3. Une phase de génération et de gestion de changement proposant différentes possibilités de changement stockées sous forme d'un *modèle d'ontologies possibles* (Model of Possible Ontologies POM). Ce modèle est utilisé pour ajouter ou supprimer des objets, mettre à jour les niveaux de pertinence et de confiance des changements, et expliquer chaque changement possible. La phase supporte un processus incrémental d'apprentissage d'ontologie (mise à jour des preuves justifiant les entités de l'ontologie en analysant les changements du corpus et, génération de suggestions et d'explications de changements d'ontologie en se basant sur de nouvelles preuves). La phase inclut aussi une gestion explicite des changements assurant la cohérence logique de l'ontologie résultant de l'apprentissage (Haase & Völker, 2008).
4. Une phase d'application de changement pour l'ajout et la suppression d'éléments de l'ontologie.

III.1.2- Identification de besoins de changement dirigée par les utilisateurs

Dans une perspective d'identification de besoins de changement dirigée par les utilisateurs, l'apprentissage d'ontologie est basé sur :

- Des procédures d'auto-évaluation formelle (Völker et al., 2005) ;
- Une précision conceptuelle basée sur l'apprentissage de disjonction (*Learning Disjointness Axioms* LeDA) (Völker et al. 2007a) et d'expressivité (*Learning Expressive Ontologies* LExO) (Völker et al., 2007b) dans le langage OWL DL.

III.1.3- Apprentissage d'ontologies cohérentes

Les connaissances extraites des corpus textuels peuvent présenter des incertitudes et être potentiellement contradictoires ce qui amène à des incohérentes logiques dans les ontologies apprises que le processus d'apprentissage résout (Haase & Völker, 2008).

L'apprentissage d'ontologie génère en premier des ontologies exprimées dans un modèle indépendant de tout langage d'ontologie : c'est le *modèle d'apprentissage d'ontologie* (Learned Ontology Model LOM). L'incertitude des connaissances est exprimée à travers les annotations de confiance associées aux éléments d'ontologie (Haase & Völker, 2008). Ensuite, une sémantique logique est introduite aux

ontologies en transformant le modèle LOM en ontologie formelle exprimée en OWL. L'algorithme de transformation tient compte des degrés de confiance pour éviter la génération d'incohérences logiques. L'objectif est d'obtenir l'ontologie cohérente qui « capture » les informations les plus certaines parmi différentes ontologies cohérentes (Haase & Völker, 2008). La sélection d'ontologie est basée sur une fonction d'évaluation considérant les degrés de confiance.

La maintenance de la cohérence est aussi basée sur les principes de résolution présentés dans (Haase & Stojanovic, 2005). Lorsque les axiomes transformés en OWL mènent à une ontologie incohérente, l'incohérence est localisée et l'axiome ayant le degré de confiance le plus faible est identifié et supprimé pour la résoudre.

III.2- Approche d'évolution d'ontologie multimédia BOEMIE[4]

L'approche BOEMIE (*Bootstrapping Ontology Evolution with Multimedia Information Extraction*) a pour objectif d'automatiser le processus d'acquisition de connaissances à partir de contenus multimédia. L'évolution d'ontologie multimédia est utilisée comme des connaissances en background, pour guider l'extraction d'information de sources multimédia, distribuées en réseau. Par ailleurs, des mécanismes de raisonnement sont appliqués aux informations sémantiques extraites afin de peupler et enrichir des ontologies.

III.2.1- Patrons de peuplement et d'enrichissement d'ontologie

BOEMIE propose une approche d'évolution basée sur des patrons. Les patrons d'évolution spécifient les inputs du processus d'évolution, c'est-à-dire les informations extraites des sources multimédia et présentées sous forme de *ABox* (instances de concepts et de relations) ; et déterminent l'opération d'évolution à exécuter (Castano et al., 2007) (Petasis, 2007): peuplement (ajout de nouvelles instances) ou enrichissement (extension par de nouveaux concepts, relations, propriétés).

Quatre patrons d'évolution sont distingués (Castano et al., 2007). Les patrons de peuplement sont utilisés lorsque l'interprétation d'une ressource multimédia (input) peut être expliquée par un seul (P1) ou de multiples (P2) concept(s) d'ontologie. Les patrons d'enrichissement sont utilisés lorsqu'il n'y a pas de concepts de l'ontologie expliquant les ressources par des métadonnées (P3) ou sans (P4). L'enrichissement d'ontologie est alors appliqué pour acquérir les connaissances manquantes. Chaque patron définit un ensemble d'activités implémentant les changements requis sous forme de mise en correspondance d'instances en peuplement, et d'apprentissage de concepts en enrichissement.

Le peuplement d'ontologie vise à identifier des instances se référant à un même objet (ou évènement). Il est basé sur la mise en correspondance d'instances et sur des techniques de clustering définies dans l'approche BOEMIE (Castano et al., 2007). L'enrichissement d'ontologie vise à apprendre de nouveaux concepts (ou relations) en appliquant des techniques de clustering; et dans certains cas (patron P3), à enrichir des concepts existants, en se basant sur des sources externes (par exemple, des ontologies de domaine ou taxonomies externes).

III.2.2- Maintenance de la cohérence

Les opérations de peuplement et d'enrichissement intègrent toutes les deux une activité de validation de la cohérence. Dans le processus de peuplement, la maintenance de la cohérence consiste à identifier et éliminer les informations redondantes et à vérifier que les instances ne causent pas de contradictions, en utilisant des services standards de raisonnement. Dans le processus d'enrichissement, la maintenance de la cohérence est limitée à la détection des incohérences liées aux modifications appliquées.

[4] http://www.boemie.org/

III.2.3- Versioning de changement

L'approche BOEMIE inclut aussi une phase de versioning et d'audit de changement permettant de coordonner l'ontologie mise à jour et rendre compte des connaissances nouvellement ajoutées ; de sauvegarder l'historique des changements appliqués ; et générer une version évoluée de l'ontologie (Castano, 2006).

III.3- Approche de gestion de changement pour des ontologies distribuées

Les travaux (Heflin et al., 1999), (Klein & Fensel, 2001), (Klein et al., 2002a), (Klein et al., 2002b), (Klein & Noy, 2003), (Maedche et al., 2003), et (Klein, 2004), traitent des mécanismes et méthodes nécessaires à la gestion des changements d'ontologie dans des environnements dynamiques et distribués. Ils proposent un formalisme de représentation de changement entre ontologies, basé sur une taxonomie d'opérations de changement définie sur le méta-modèle OWL. Ils proposent également une méthode pour l'interprétation des données issues de différentes versions d'ontologie.

La gestion de changement est assurée à travers un Framework global permettant de générer des informations sur les changements, de dériver des informations sur des changements additionnels et de résoudre des problèmes spécifiques (Klein, 2004). Différentes méthodes et techniques de gestion de changement sont proposées (Klein, 2004): des algorithmes de comparaison, des mappings d'ontologies, des services de raisonnement, une validation humaine, la visualisation des changements, des heuristiques de prédiction d'effets, et des lignes directrices pour certaines tâches spécifiques.

III.3.1- Gestion distribuée des changements

L'étude fait ressortir trois caractéristiques spécifiques à la gestion de changements dans des environnements distribués. Ces caractéristiques ne sont liés qu'à la nature de l'ontologie et ne dépendent pas des paramètres de distribution (Klein et al., 2002b), (Klein, 2004) :

- La propagation d'un changement à d'autres niveaux d'interprétation, dépend du fait que le changement modifie ou non la spécification ou la conceptualisation de l'ontologie ;
- Etant distribuées, les ontologies peuvent être utilisées pour différentes tâches. La maintenance de la cohérence ne peut être focalisée sur une seule caractéristique à préserver. Les conséquences de changements doivent ainsi être étudiés spécifiquement à des cas d'utilisation de l'ontologie ;
- L'expressivité et la sémantique du langage d'ontologie peuvent parfois être exploitées pour résoudre des problèmes spécifiques de gestion de changement comme par exemple, la mise en correspondance de versions de changements.

III.3.2- Ontologie d'opérations de changement et langage de spécification de changement

L'approche souligne l'importance de la spécification des changements dans l'échange d'informations de changement entre utilisateurs, outils ou processus. La spécification des changements est décrite par un ensemble d'opérations de changements dérivées du méta-modèle du langage d'ontologie et/ou des opérations de changement composites qui prescrivent les étapes à suivre pour transformer une version antérieure d'une ontologie en une nouvelle version (Klein, 2004). La spécification des changements inclut également des relations conceptuelles et des relations d'évolution entre anciennes et nouvelles versions de constructions ontologiques (parties de l'ontologie), des métadonnées sur les changements et les conséquences des changements.

Une ontologie d'opérations de changement ainsi qu'un langage de spécification de changement (basé sur RDF) ont été définis. L'ontologie d'opérations de changement étend la taxonomie des changements OWL

(voir section II.3.3) à un langage générique de spécification de changement. Ce langage permet de spécifier les transformations d'une version d'ontologie en une autre version. Il assure (Klein, 2004):

- Des descriptions de changement concises et à différents niveaux de granularité ;
- La possibilité de révocation ou réexécution de changement ;
- L'analyse des effets de changement ;
- L'interopérabilité entre outils sur la base d'informations de changement formelles et non ambigües ;
- L'expressivité des descriptions de la portée des modifications.

III.4- Processus global d'évolution d'ontologie KAON

Dans (Stojanovic, 2004) un processus global d'évolution d'ontologie est présenté. Ce processus assure la spécification de la sémantique des changements, la maintenance de la cohérence et la propagation des changements pour les ontologies KAON. L'évolution d'ontologie est définie comme étant l'interprétation formelle de tous les besoins de changement, identifiés de diverses sources ; l'application des changements et leur propagation aux artefacts dépendants tout en préservant la cohérence (Stojanovic, 2004). Les artefacts dépendants incluent les objets référencés par l'ontologie évoluée ainsi que les ontologies et applications dépendantes.

Une infrastructure de gestion d'ontologies KAON, principalement développée pour des ontologies business, implémente le processus global à travers une méthodologie d'évolution à six phases (Stojanovic, 2004), (Oberle et al., 2004).

III.4.1- Phase d'identification de changement

La phase du lancement du processus consiste à identifier les changements d'ontologie à appliquer. Elle est basée sur des besoins de changement explicites déterminés par l'application de méthodes d'identification dirigées par les données et l'usage.

Les changements dirigés par les données (*Data-driven Change*) sont dérivés des instances de l'ontologie en appliquant des techniques de datamining, d'analyse de concepts formels et différentes heuristiques (Stojanovic, 2004) (Maedche et al., 2003).

Les changements dirigés par l'usage (*Usage-driven Change*) sont basés sur des patrons d'usage d'ontologie (*ontology usage patterns*) dérivés de l'analyse du comportement des utilisateurs et de la traçabilité des requêtes appliquées à l'ontologie. Ils permettent d'identifier les parties les plus utilisées de l'ontologie (Stojanovic et al., 2003a).

III.4.2- Phase de représentation de changement

Cette phase a pour objectif de décrire les changements identifiés selon les spécifications du langage KAON. Un changement peut être représenté selon trois niveaux de granularité : changement élémentaire, changement composite et changement complexe (voir section II.3.3).

III.4.3- Phase de sémantique de changement

Faire évoluer une ontologie suppose la préservation de sa cohérence afin qu'elle garde sa pertinence par rapport au domaine qu'elle modélise et aux applications qui l'utilisent (Haase & Völker, 2005). La phase de sémantique de changement a pour objectif d'évaluer et résoudre les effets de changement de manière systématique en assurant la cohérence de toute l'ontologie (Stojanovic et al., 2002a). La cohérence de l'ontologie (*ontology consistency*) est définie comme suit (Stojanovic, 2004, p. 30): « une ontologie est cohérente par rapport à son modèle – d'ontologie – si et seulement si, elle satisfait les contraintes spécifiées par ce modèle ». L'approche globale est dédiée aux ontologies KAON. Cependant, dans (Haase

& Stojanovic, 2005) (Haase et al., 2005), les auteurs ont décrit les sémantiques de changement d'évolution d'ontologies OWL.

III.4.3.1- Maintenance de la cohérence des ontologies KAON

KAON est un langage centré contraintes, basé sur l'hypothèse du monde fermé. Le modèle de cohérence est défini à travers un ensemble de contraintes décrivant les invariants du modèle KAON (distinction d'identité, hiérarchie de concept, fermeture de concept, fermeture de hiérarchie de concept, etc.), des contraintes logicielles et des contraintes utilisateurs (Stojanovic et al., 2003b) (Stojanovic, 2004). Les contraintes décrivant les invariants du modèle doivent être obligatoirement satisfaites. Les contraintes logicielles peuvent être temporairement invalides afin de faciliter le peuplement d'ontologie par exemple. Les contraintes utilisateurs représentent plutôt des directives pour la construction d'ontologies bien formées.

Deux types d'incohérence sont distingués: incohérence structurelle et incohérence sémantique. L'incohérence structurelle se rapporte au non respect des contraintes du modèle KAON. L'incohérence sémantique altère la signification des entités de l'ontologie. Seule l'incohérence structurelle est considérée vu qu'elle peut être vérifiée avec l'assistance de l'ingénieur d'ontologie. La cohérence sémantique par contre, se base sur des informations sémantiques qui ne sont pas explicitement exprimées dans le modèle standard de l'ontologie KAON et donc, pas facilement contrôlables (Stojanovic, 2004).

La maintenance de la cohérence est assurée à travers trois étapes : 1) la localisation des incohérences (les sous-ensembles incohérents minimaux), 2) la détermination des résolutions possibles, et 3) la sélection et l'application du meilleur changement (le changement requis et ses changements dérivés qui le résolvent).

Vérification de la cohérence. Deux approches de vérification de cohérence sont présentées (Stojanovic, 2004):

- Vérification *a posteriori* : une seule vérification est exécutée pour tous les changements appliqués ;
- Vérification *a priori* : la vérification est faite avant l'application d'un changement. A Chaque changement est associé un ensemble de pré-conditions à satisfaire pour qu'il puisse être appliqué. Par ailleurs, avant toute application de changement, l'ontologie est supposée cohérente.

La vérification *a posteriori* est une approche très coûteuse comme elle est appliquée à l'ontologie dans son ensemble et qu'elle implique, pour la résolution, des mécanismes de retours arrière. De plus, elle ne permet pas d'expliquer les impacts de changement et de préciser lequel(s) de(s) changement(s) appliqué(s) a causé les incohérences puisque les changements sont appliqués en batch. Pour délimiter la vérification à la portée du changement et ne pas avoir à faire des retours arrière pour remettre l'ontologie à son état cohérent, c'est la seconde approche qui a été adoptée dans (Stojanovic, 2004). La vérification *a priori* est basée sur la spécification des pré-conditions nécessaires à satisfaire pour assurer l'applicabilité d'un changement. De plus, un ensemble de post-conditions à vérifier après application d'un changement est défini, sa satisfaction permet de valider le changement.

Résolution des incohérences. Deux approches sont proposées pour une résolution automatique des incohérences (Stojanovic et al., 2003b) (Stojanovic et al., 2002a) (Stojanovic, 2004):

1. Approche procédurale : la maintenance se base sur les contraintes du modèle de cohérence et sur un ensemble prédéfini de règles à appliquer pour les satisfaire. L'approche est organisée en deux phases principales :

- Prise en charge de la sémantique du changement: le changement requis est décomposé en une séquence de changements, appliqués un à un. A chaque changement, les pré-conditions associées sont vérifiées puis les résolutions d'incohérences sont générées. Différentes possibilités de résolution appelées *stratégies d'évolution*, sont générées pour un changement. Chacune propose un ensemble additionnel de changements résolvant l'incohérence en tenant compte d'un ensemble de besoins particuliers de l'ingénieur d'ontologie. L'objectif de générer plusieurs stratégies est d'adapter la politique d'évolution aux différentes applications de l'ontologie. Le processus est itéré jusqu'à ce qu'il n'y ait plus de changement à prendre en charge,
- Application du changement : tous les changements sont appliqués à l'ontologie tout en considérant les post-conditions qui leurs sont associées

2. Approche déclarative : la maintenance se base sur un ensemble complet d'axiomes inférés, formalisant l'évolution. L'approche est organisée en trois phases :
   - Formalisation de la requête: l'ingénieur d'ontologie spécifie sa demande de changement de façon déclarative à travers une collection de changements d'ontologie composée de deux ensembles de changements : *changements à appliquer* (supprimer le concept A, par exemple) et *changements à ne pas appliquer* (ne pas supprimer le concept B, par exemple),
   - Résolution du changement : seul le premier ensemble de changements est appliqué afin de détecter les incohérences causées. Ensuite, vient la génération des résolutions qui tient compte des deux ensembles de changements. Tous les changements possibles résolvant les incohérences sont générés. La résolution est ensuite réduite à un problème de graphe de recherche où les nœuds correspondent aux ontologies modifiées et les arêtes aux changements appliqués. La recherche est guidée par les contraintes de l'ingénieur d'ontologie, les règles du modèle de cohérence et les annotations associées aux nœuds et arêtes du graphe,
   - Tri des solutions : toutes les ontologies cohérentes possibles sont classées sur la base des méta-informations fournies par l'ingénieur d'ontologie.

Il a été démontré qu'aussi bien l'approche procédurale que l'approche déclarative, permettent d'appliquer le même ensemble de changements et offrent les mêmes possibilités de contrôle et de d'adaptation des résolutions. Leur comparaison ne peut que se baser sur des critères subjectifs tels que l'efficacité du système d'évolution qui les implémente (Stojanovic, 2004).

III.4.3.2- Maintenance de la cohérence des ontologies OWL

OWL est un langage centré axiome, basé sur l'hypothèse du monde ouvert. Quatre approches différentes ont été proposées pour prendre en charge les incohérences OWL (Haase et al., 2005): évolution d'ontologies cohérentes, réparation des incohérences, raisonnement en présence d'incohérences, et raisonnement multi-versions. Dans (Haase & Stojanovic, 2005), un modèle formel a été proposé pour l'évolution d'ontologies OWL cohérentes. La cohérence est définie par un ensemble de conditions à satisfaire et est classée en trois niveaux : cohérence structurelle, cohérence logique et cohérence utilisateur.

- La cohérence structurelle se réfère aux contraintes du langage de l'ontologie ;
- La cohérence logique se réfère à la sémantique formelle de l'ontologie et à sa satisfiabilité. Elle vérifie que l'ontologie est sémantiquement correcte et ne présente pas de contradictions logiques ;

- La cohérence utilisateur se réfère à l'usage de l'ontologie et son contexte d'application. Les contraintes sont explicitement définies par l'utilisateur. Deux types de contraintes sont considérés (Haase & Stojanovic, 2005):
    - Cohérence générique spécifiant des critères qualificatifs de modélisation d'ontologie tels que l'application des méta-propriétés OntoClean (Guarino & Welty 2002),
    - Cohérence dépendante du domaine spécifiant des conditions particulières liées au domaine modélisé et dépendant de l'expertise de l'utilisateur.

La définition de la sémantique de OWL se base sur un modèle théorique associant la syntaxe de OWL au modèle du domaine modélisé par l'ontologie (ses contraintes) : satisfaire une ontologie par une interprétation est contraint par la satisfaction de tous ses axiomes (Haase & Stojanovic, 2005).

Vérification de la cohérence. La cohérence n'est vérifiée que pour des changements d'ajout d'axiomes, ce choix étant basé sur la monotonicité logique de OWL. L'objectif de la vérification est de localiser la sous-ontologie incohérente minimale soit un ensemble minimal de d'axiomes contradictoires (Haase & Stojanovic, 2005).

Résolution des incohérences. Pour les incohérences structurelles, une résolution à base de règles de réécriture est proposée afin de transformer les axiomes en expressions compatibles avec les contraintes du modèle OWL Lite (Haase & Stojanovic, 2005). Pour la résolution des incohérences logiques, des stratégies de résolution d'incohérences sont introduites, elles se basent sur les contraintes de OWL Lite (Haase & Stojanovic, 2005). Chaque condition de cohérence est reliée à une fonction de résolution spécifiant les changements additionnels à appliquer. Ces changements correspondent à un ensemble d'axiomes à supprimer pour aboutir à une ontologie cohérente (cohérence logique) tout en minimisant l'impact sur l'ontologie existante. Pour sélectionner les axiomes à supprimer, les auteurs (Haase & Stojanovic, 2005) proposent l'idée de définir des fonctions de sélection en s'inspirant des travaux sur l'incertitude et la réification et sur les recherches en ontologies contextuelles pour dégager des métadonnées décrivant les axiomes (confidence, degré de certitude, caractéristiques contextuelles, etc.). Une sous-ontologie cohérente maximale est obtenue (Haase & Stojanovic, 2005). Les changements additionnels sont présentés à l'expert afin qu'il détermine quels sont les changements à retenir.

III.4.4- Phase de propagation de changement

Dans la continuité du processus global d'évolution d'ontologie et après la résolution des incohérences, la phase de propagation assure comme son nom l'indique, la propagation des changements de l'ontologie aux artefacts dépendants afin de préserver la cohérence globale. Ces artefacts peuvent être des ontologies réutilisées par l'ontologie évoluée ou des applications distribuées. La propagation assure le suivi et la diffusion des changements appliqués. Plusieurs approches de synchronisation sont proposées et détaillées dans (Maedche et al., 2003) (Stojanovic, 2004).

III.4.5- Phase d'implémentation de changement

Cette phase consiste à implémenter physiquement le changement requis et les changements dérivés qui le résolvent. Les changements sont d'abord notifiés à l'ingénieur d'ontologie pour être approuvés, puis appliqués. Tous les changements effectués sont enregistrés afin de faciliter leur révocation si besoin.

L'archivage des changements KAON est basé sur deux notions spécifiques (Stojanovic, 2004): l'*ontologie d'évolution* et le *journal d'évolution.* L'ontologie d'évolution définit un modèle des changements applicables à une ontologie. Le journal d'évolution – basé sur le modèle formel de

l'ontologie d'évolution – décrit l'historique des changements appliqués à travers des séquences d'informations chronologiques sur les changements. Il renferme des connaissances sur le développement et la maintenance de l'ontologie de domaine. La modélisation des changements (ontologie d'évolution) et de l'historique de leur application (journal d'évolution) facilite la synchronisation de l'évolution et facilite l'annulation des changements si besoin.

III.4.6- Phase de validation de changement

C'est la phase de validation finale de tous les changements. Elle assure la réversibilité des changements s'ils sont finalement désapprouvés par l'utilisateur, la justification des changements et de leur utilisabilité (Stojanovic, 2004).

A ce stade, d'autres problèmes peuvent être identifiés, inférant de nouveaux besoins de changement et relançant un autre cycle du processus d'évolution.

III.5- Approche d'évolution d'ontologie basée sur les principes de changement de croyance

La motivation principale de ces travaux (Flouris et al., 2005) (Flouris & Plexousakis, 2006) (Flouris et al., 2006a) (Flouris, 2006) était que la maturité des recherches en changement de croyance pourrait être profitable à l'évolution d'ontologie et leur apporter la formalisation nécessaire. Les changements de croyance se réfèrent à l'adaptation automatique d'une base de connaissances, à de nouvelles connaissances et ce, sans intervention humaine dans le processus (Flouris et al., 2006a). L'hypothèse de départ de ces travaux était que l'adoption des principes et algorithmes de changement de croyance pourrait réduire la dépendance à l'ingénieur de connaissances, dans un processus d'évolution d'ontologie. L'approche est destinée à des applications basées sur des ontologies changeant fréquemment et à des applications autonomes telles que les agents logiciels ; c'est-à-dire lorsqu'il est très difficile de gérer manuellement ou de manière semi-automatique les changements.

La théorie des changements de croyance considérée par ces travaux est la théorie AGM initiée par les trois auteurs Alchourron, Gärdenfors et Makinson (1985) et appliquée à la révision des croyances comme une méthode spécifiant les propriétés minimales qu'un processus de révision doit avoir.

III.5.1- Champ d'application de l'approche

L'étude montre que la théorie AGM ne peut être appliquée qu'aux logiques classiques telles que la logique propositionnelle (*Propositional Logic*) et la logique de premier ordre (*First-Order Logic*). Elle ne peut être (directement) appliquée aux standards de représentation d'ontologie tels que les logiques de description (*Description Logics*) et OWL (Flouris, 2006). Les techniques de changement de croyance ne sont pas applicables aux logiques de description sous l'hypothèse d'un monde fermé. Et sous l'hypothèse du monde ouvert, elles ne sont applicables que pour certaines logiques de description mais pas en OWL (Flouris et al., 2005).

III.5.2- Opérations d'évolution d'ontologie

Quatre opérations d'évolution d'ontologie sont distinguées en se basant sur la littérature des changements de croyance (Flouris & Plexousakis, 2006): révision d'ontologie, contraction d'ontologie, mise à jour d'ontologie et effacement d'ontologie (ontology erasure). La révision et la contraction d'ontologie reflètent un changement dans la perception du domaine modélisé c'est-à-dire sa conceptualisation (état statique du monde) ; la mise à jour et l'effacement d'ontologie reflètent un changement dans le domaine lui-même.

Ces quatre opérations, étant basées sur des paradigmes différents, ne correspondent pas vraiment aux opérations de changement utilisées dans la littérature d'évolution d'ontologie. Inspirées des principes de changement de croyance, elles donnent une vision différente sur la manière d'interpréter et de gérer un changement. Ces opérations sont centrées « faits » (fact-centered) (Flouris, 2006): chaque nouveau fait représente un certain besoin pour l'évolution d'ontologie. L'opération impliquée dans un changement, est déterminée par l'identification du type du nouveau fait (état du monde statique ou dynamique) et de ses impacts (ajout ou suppression de connaissances). Ainsi, le système identifie les modifications nécessaires et les exécute automatiquement. Les approches standards, elles, sont centrées modification (modification-centered) (Flouris, 2006): elles tiennent compte des modifications qui doivent être physiquement appliquées pour répondre à un nouveau fait, ce qui rend le processus de gestion de changement moins compliqué et donne à l'ingénieur d'ontologie plus de contrôle.

III.5.3- Algorithme d'évolution d'ontologie

Le processus d'évolution d'ontologie est réalisé par un algorithme d'évolution d'ontologie prenant en entrée, une ontologie et un changement représentés par un ensemble d'axiomes et les appliquant à une ontologie. L'algorithme se rapproche de l'approche axiomatique décrite dans (Haase & Stojanovic, 2005). Les opérations d'évolution – révision, contraction, mise à jour et effacement – sont implémentées en quatre fonctions d'évolution. Dans (Flouris, 2006), un premier algorithme (totalement automatique) de contraction d'ontologies basées sur des logiques de description compatible AGM, est introduit. Cependant, l'algorithme tient compte uniquement d'aspects syntaxiques.

Appréhender l'évolution d'ontologie selon une perspective de révision de croyance est assez différent des approches courantes. En effet, ces travaux admettent que les fondements sont différents (Flouris & Plexousakis, 2006) (Flouris, 2006): la révision de croyance est basée sur des méthodes de postulats alors que les approches d'évolution d'ontologie reposent sur des constructions explicites impliquant la participation de l'ingénieur d'ontologie pour faire face à des problèmes techniques et pratiques de gestion de changement. L'application de la théorie AGM à l'évolution d'ontologie est proposée comme une approche complémentaire pour pallier l'absence de formalisation efficace du processus, dans les approches actuelles d'évolution d'ontologie.

III.5.4- Maintenance de la cohérence

La maintenance de la cohérence a été prise en compte dans cette approche. Deux notions sont distinguées (Flouris, 2006) : *consistency* et *coherence* pour lesquelles une seule traduction existe en français – la cohérence – bien que le sens soit tout à fait différent. La cohérence en tant que *consistency* est liée à la satisfaction de tous les axiomes (en logique de description) de l'ontologie. Ceci correspond au sens adopté dans ce manuscrit pour la cohérence logique. La cohérence en tant que *coherence* est liée à la satisfaction de contraintes prédéfinies qui relèvent d'une bonne conception d'ontologie. Cette cohérence, étant liée à des problèmes de conception, n'est pas prise en compte dans l'algorithme d'évolution. La maintenance de la cohérence – logique – est basée sur l'application des principes de changement de croyance (Flouris, 2006).

III.6- Approche de détection de changement basée sur l'historique des versions

Dans (Plessers & De Troyer, 2005) (Plessers et al., 2007), une approche de détection de changement a été proposée pour un Framework d'évolution d'ontologie OWL DL. L'approche vise à détecter des changements qui n'ont pas été explicitement demandés par l'ingénieur d'ontologie, et à générer – automatiquement – une vue globale détaillée des changements appliqués en se basant sur un ensemble de

définitions de changements. Différentes vues de changement peuvent être fournies pour une même évolution puisque chaque utilisateur peut avoir son propre ensemble de définitions de changement.

Un langage de définition de changement CDL (*Change Definition Language*) a été défini (Plessers et al., 2007). Il fournit différents niveaux d'abstraction et permet aux utilisateurs de spécifier formellement la signification des changements. Les changements sont exprimés sous forme de requêtes temporelles appliquées à l'historique des versions. L'historique des versions permet de sauvegarder pour chaque concept défini dans l'ontologie, toutes ses versions.

III.6.1- Aperçu sur le Framework d'évolution

Le Framework proposé prend en considération deux possibles tâches d'évolution, chacune ciblant un rôle utilisateur : *évolution à la demande* (Evolution-on-request) pour les ingénieurs d'ontologie modifiant l'ontologie, et *évolution en réponse* (*Evolution-in-response*) pour les « mainteneurs » des artefacts dépendants cherchant des informations sur les changements appliqués. L'approche de détection de changement est appliquée dans ces deux tâches d'évolution mais à des étapes différentes.

III.6.1.1- Evolution à la demande

Cette tâche est organisée en cinq phases : (1) d'abord, l'ingénieur d'ontologie exprime sa demande de changement dans le langage CDL (voir section III.6.2) ; (2) ensuite vient la phase de maintenance de la cohérence permettant de localiser les incohérences et les résoudre (voir section III.6.4) ; (3) la phase de détection de changement intervient à ce niveau pour identifier les changements qui ont eu lieu en conséquence des changements de départ ; (4) elle est suivie de la phase de recouvrement de changement permettant d'éliminer les changements intermédiaires qui ne sont pas nécessaires (Plessers, 2006) ; (5) et au final, la phase d'implémentation de changement permet d'implémenter le changement appliqué à une copie locale de l'ontologie, dans la version publique.

III.6.1.2- Evolution en réponse

Tenant compte du fait que les « mainteneurs » des artefacts dépendants peuvent définir différemment les changements appliqués par l'ingénieur d'ontologie, cette tâche leur permet d'approuver ou non les changements appliqués et de décider de leur propagation ou non aux artefacts dépendants. L'évolution en réponse est organisée en trois phases : (1) la phase de détection de changement intervient en premier pour générer l'historique d'évolution (voir section III.6.3) ; (2) ensuite la phase coût d'évolution permet d'évaluer le coût de la mise à jour des artefacts dépendants ; (3) et finalement, si la mise à jour est approuvée, la phase de cohérence de version assure la cohérence des artefacts dépendants avec la version évoluée de l'ontologie. Autrement, les artefacts dépendants restent inchangés et cohérents avec l'ancienne version de l'ontologie.

III.6.2- Langage de définition de changement

Le langage de définition de changement CDL (Change Definition Language) permet de spécifier de manière formelle et déclarative les définitions des changements (différences entre l'ancienne version et la version courante de l'ontologie). Basé sur la logique temporelle, il définit un changement d'ontologie en termes de pré-conditions et post-conditions. La syntaxe et la sémantique de langage CDL sont détaillées dans (Plessers et al., 2007).

III.6.3- Historique d'évolution

Les historiques d'évolution ont pour objectif d'exprimer les différentes interprétations qu'ont les utilisateurs de l'évolution de l'ontologie. Durant la phase de détection de changement, l'application d'une requête temporelle sur un historique de version permet de générer une collection de définitions de changement – exprimées dans le langage CDL – correspondant aux occurrences des changements appliqués. Cette collection constitue un historique d'évolution. L'historique de version sauvegarde l'historique d'évolution de chaque entité définie dans l'ontologie. C'est la ligne de vie de l'entité, elle commence à sa création, décrit ses différentes modifications et continue jusqu'à la fin de son cycle de vie (sa suppression).

La représentation des historiques de versions se base sur une approche par *snapshot* qui capture les différents états d'une ontologie dans le temps et sauvegarde la trace d'évolution de chaque concept de l'ontologie (Plessers et al., 2007).

III.6.4- Maintenance de la cohérence

Dans (Plessers & De Troyer, 2006), les auteurs ont définit une approche et un algorithme de localisation des axiomes causant les incohérences et ont proposé un ensemble de règles que l'ingénieur d'ontologie peut appliquer pour résoudre les incohérences. Le modèle de cohérence se base sur les contraintes de OWL DL.

La vérification de la cohérence concerne deux scenarii de changement (Plessers & De Troyer, 2006):

- Ajouter/modifier des axiomes au niveau terminologique de OWL DL, la *TBox* (classes et propriétés) : la satisfiabilité de la TBox est d'abord vérifiée, ensuite la cohérence de la *ABox* par rapport à la *TBox* ;
- Ajouter/modifier des axiomes au niveau assertionnel de OWL LD, la *ABox* (instances) : dans ce cas la vérification n'est appliquée qu'à la *ABox*.

Compte tenu de la monotonicité logique de OWL DL, la vérification de la cohérence n'est pas appliquée après une opération de suppression.

L'algorithme de sélection des axiomes causant les incohérences est basé sur deux types d'arbres de localisation (*tracking trees*) (Plessers & De Troyer, 2006) : les arbres de transformation d'axiomes permettant de garder la traçabilité des transformations d'axiomes qui se sont produites à l'étape de prétraitement de l'algorithme et les arbres de dépendance de concepts permettant de garder la traçabilité des axiomes ayant engendré des conflits durant l'exécution de l'algorithme. Sur la base des informations sur les conflits produits, les arbres de transformation d'axiomes et les arbres de dépendance de concepts, l'algorithme détermine l'ensemble des axiomes causant les incohérences.

L'hypothèse stipulée pour la résolution des incohérences est de considérer une incohérence comme une conséquence de trop d'axiomes restrictifs – contradictoires les uns des autres – devant être diminués (*to be weakened*) (Plessers & De Troyer, 2006).

Les auteurs proposent une collection de règles guidant l'ingénieur d'ontologie dans la résolution des incohérences. Une règle peut soit faire appel à une autre règle soit appliquer un changement à un axiome. Certaines règles peuvent être appliquées pour diminuer des axiomes (elles décrivent par exemple, comment diminuer la définition d'un concept ou l'inclusion d'un concept) et d'autres, pour diminuer ou augmenter des concepts (relations de disjonction, quantification existentielle, etc.).

III.7- Approche de gestion de l'évolution d'un web sémantique d'entreprise

Dans (Luong, 2007), une approche de gestion d'évolution d'un web sémantique d'entreprise est présentée. Les ontologies et les annotations y sont considérées en tant que composants principaux d'un web sémantique d'entreprise. L'approche se focalise principalement sur la propagation des changements d'ontologie aux annotations sémantiques. Elle récupère les historiques d'évolution fournis par l'éditeur d'ontologie ECCO[5] (Editeur Collaboratif et Contextuel d'Ontologie) sous forme d'un journal d'évolution et analyse et répare les annotations sémantiques sur leur base.

L'approche supporte aussi l'évolution d'ontologie sans trace de changement. Elle permet de travailler sur plusieurs versions d'une ontologie, de les comparer pour faire identifier les entités qui existent dans l'une et pas dans l'autre et propose à l'utilisateur les différentes possibilités de correction des incohérences des annotations.

Le processus d'évolution est organisé en trois étapes : la représentation des changements, la résolution des changements et la propagation des changements aux annotations.

III.7.1 Phase de représentation des changements

Cette phase permet de décrire formellement la syntaxe, les paramètres et la sémantique d'un changement, de même que les conditions à satisfaire avant et après son application et ce, à travers une *ontologie d'évolution*. L'ontologie d'évolution se base sur un modèle générique de description d'ontologie, inspiré des travaux de (Stojanovic, 2004), définissant une ontologie sous forme d'un *tuple* composé de : (concepts, propriétés, attributs, instances, hiérarchies de concepts, hiérarchies de propriétés, fonction qui rend le domaine d'une propriété, fonction qui rend le co-domaine d'une propriété et une fonction qui rend le nom d'une entité de l'ontologie.). Elle formalise les changements d'ontologies exprimées en RDF(S).

Les changements modélisés correspondent à des changements élémentaires (affectant une seule entité de l'ontologie) et des changements composites (affectant des entités d'un même voisinage), selon la classification de (Stojanovic, 2004). Outre le niveau d'abstraction, les changements modélisés sont aussi classés selon leur impact sur les annotations sémantiques. Deux types de changements se dégagent : des changements entrainant une correction obligatoire des annotations (pour éliminer les incohérences causées) et des changements entrainant des corrections facultatives (pas d'incohérences causées) (Luong, 2007).

L'ontologie d'évolution modélise également les traces des changements effectués et des annotations sémantiques mises à jour.

III.7.2 Phase de résolution des changements

L'approche s'intéressant principalement à l'impact des changements sur les annotations sémantiques référençant les entités de l'ontologie évoluée (instances, concepts et propriétés), la maintenance de la cohérence tient compte uniquement du niveau structurel. Par ailleurs, il n'y pas une réelle phase de maintenance de l'ontologie évoluée avec détection et résolution des incohérences. En effet, les changements sont censés être appliqués en amont dans le système ECCO c'est pourquoi, seulement des propositions de résolutions ont été définies avec la perspective de les implémenter dans ECCO (Luong, 2007). Ces propositions consistent en une bibliothèque de résolutions associant des stratégies de résolution aux différents changements modélisés par l'ontologie d'évolution. Elle s'inspire des stratégies de résolution proposées dans (Stojanovic, 2004).

Par contre, pour préparer la résolution des incohérences au niveau des annotations sémantiques (voir section III.7.3), cette phase inclut une activité de comparaison de versions d'ontologie. La comparaison de

[5] http://www-sop.inria.fr/edelweiss/projects/ewok/publications/ecco.html

deux versions de l'ontologie se base sur l'intégration du moteur de recherche sémantique CORESE[6] (COnceptual REsource Search Engine) (Corby et al. 2004) permettant de lancer des requêtes sur les deux versions pour rechercher des informations sur les hiérarchies de concepts et de propriétés ainsi que sur les annotations qui les utilisent. La comparaison des résultats du moteur sur les deux versions permet de déterminer les différences de structure entre elles. Cette tâche est facilitée si la traçabilité des changements appliqués a été assurée en amont. Dans ce cas, en se basant sur la trace de changement fournie par l'éditeur ECCO utilisé en interfaçage, la fonction de comparaison permet de préciser non seulement les changements effectués, les entités concernées et les annotations affectées mais aussi des informations supplémentaires décrivant comment les changements ont été réalisés.

III.7.3 Phase de propagation des changements

La propagation des changements aux annotations a pour objectif de maintenir la cohérence des annotations par rapport aux entités de l'ontologie qu'elles référencent. Tout comme pour les changements, la résolution des incohérences des annotations se base sur un modèle formel de spécification des annotations, des contraintes de cohérence et des stratégies de résolution.

Le modèle d'annotation sémantique –basé sur le modèle RDF– est décrit par un *tuple* composé de (ressources, noms de concepts, noms de propriétés, valeurs littérales et de triplets RDF).

La maintenance des annotations se base sur la vérification de contraintes exprimant comment référencer de manière cohérente les termes définis dans le vocabulaire de l'ontologie (Luong, 2007).

Les stratégies de résolution d'incohérence d'annotations sémantiques sont définies sur la base des stratégies de résolution de changement. Elles sont classées selon trois types d'opérations (Luong, 2007): des stratégies de non changement (les triplets sont conservés), des stratégies de remplacement (remplacement d'éléments de triplets par d'autres disponibles et convenables), et des stratégies de suppression (suppression de triplets entiers).

Deux approches de résolution sont proposées selon que l'évolution de l'ontologie est gérée en gardant la trace des changements ou non (Luong, 2007):

- Une approche procédurale employée lorsque le système dispose des traces de changement, permet d'identifier les annotations incohérentes à partir des informations enregistrées et les résoudre en appliquant des procédures de correction. La résolution est automatique, des stratégies par défaut sont générées et appliquées systématiquement. Un mode semi-automatique est aussi prévu, donnant la main à l'utilisateur pour choisir parmi les suggestions de modification, celle qui lui convient le mieux;
- Une approche basée sur des règles employée lorsque le système ne dispose pas de traces de changement, permet d'identifier les annotations contenant des références obsolètes en se basant sur les contraintes de cohérence (règles de détection) et de les réparer (règles de correction). Si l'identification des incohérences est conduite automatiquement, la correction quant à elle, se fait semi-automatiquement en proposant à l'utilisateur –sans chercher à détecter les changements appliqués à l'ontologie– les « meilleures manières » de corriger les annotations. Un mécanisme de sélection des entités les plus pertinentes de l'ontologie évoluée, à associer aux annotations pour remplacer celles qui sont devenues obsolètes, est développé.

Après la présentation des approches existantes en évolution d'ontologie, nous décrivons dans la section suivante, les principaux outils supportant des fonctionnalités d'évolution d'ontologie dont certains implémentant ces approches.

[6] http://www-sop.inria.fr/edelweiss/software/corese/

IV- OUTILS SUPPORTANT L'EVOLUTION D'ONTOLOGIE

Gérer manuellement les changements d'une ontologie est une tâche complexe et coûteuse. L'ingénieur d'ontologie doit contrôler tous les effets de changement, les résoudre et évaluer l'impact du changement sur l'ontologie. Il y a besoin d'outils appropriés offrant les moyens techniques nécessaires à l'évolution d'ontologie.

L'évolution d'ontologie doit faire partie des fonctionnalités d'un éditeur d'ontologie pour guider le développement d'ontologie selon un processus itératif et dynamique. Pourtant les besoins en évolution d'ontologie ne sont pas considérés par tous les éditeurs d'ontologie existants. Ces besoins sont principalement (Stojanovic, 2004): des besoins fonctionnels, des besoins de personnalisation, des besoins de transparence, des besoins de réversibilité, des besoins d'audit, des besoins de raffinement et des besoins de d'utilisabilité. Dans (Noy et al., 2006), les besoins fonctionnels considérés particulièrement pour un environnement collaboratif, sont : l'annotation des changements, historique des changements de concept, la représentation des changements d'une version de l'ontologie à une autre, la définition des privilèges d'accès, l'application de requêtes sur une ancienne version en utilisant le vocabulaire d'une nouvelle version et la génération de synthèses de changement. D'autres besoins fonctionnels spécifiques à l'édition collaborative asynchrone, l'édition continue, l'édition laconique et l'édition non-supervisée sont aussi présentés.

Outre certains éditeurs d'ontologie existants supportant certains aspects d'évolution, les travaux courants en évolution d'ontologie – dont certaine décrits dans les sections précédentes – proposent des outils spécialisés dont l'objectif est soit de guider l'utilisateur à appliquer manuellement les changements soit de gérer et appliquer les changements automatiquement. Certains des ces outils permettent des éditions collaboratives (Duineveld et al., 2000) (Noy et al., 2006) d'autres, supportent des aspects liés au versioning d'ontologie (Duineveld et al., 2000) (Klein et al., 2002a) (Noy & Musen, 2002) (Noy et al., 2006).

IV.1- Outil KAON[7]

Le Framework KAON proposé par l'Université de Karlsruhe, implémente un système d'évolution d'ontologie (Stojanovic, 2004). Outre l'automatisation du processus d'évolution, la boite à outils KAON guide l'ingénieur d'ontologie pour formuler ses besoins de changement en fournissant des informations supplémentaires et des suggestions d'amélioration de l'ontologie. Elle offre une fonctionnalité d'identification de besoins de changement dirigée par les données (Maedche et al., 2003), permet aux utilisateurs de définir des « stratégies d'évolution » contrôlant l'application des changements et, facilite l'adaptation de l'ontologie aux besoins des utilisateurs finaux (identification des besoins dirigée par l'usage) (Stojanovic et al., 2002a).

IV.2- Editeur Protégé[8]

Protégé est outil développé par le groupe d'informatique médicale de Stanford. Il regroupe un éditeur d'ontologie, open source, et un Framework de bases de connaissances. Protégé est doté d'un environnement graphique et interactif de conception d'ontologie et d'acquisition de connaissances. Son architecture est orientée composants facilitant l'enrichissement des fonctionnalités de l'éditeur par l'ajout de plugins (tels que le plugin Protégé-OWL). Certains plug-ins sont spécifiques à des aspects d'évolution d'ontologie et sont présentés dans les sections ci-dessous.

[7] http://kaon.semanticweb.org
[8] http://protege.stanford.edu/

IV.3- Outils de Versioning d'Ontologie

Dans le Framework de gestion de changement proposé pour les ontologies distribuées (voir section III.3), plusieurs prototypes spécialisés ont été développés (Noy & Klein, 2003) (Noy et al., 2004) (Klein, 2004) :

- L'outil OntoView implémente une procédure de détection de changement pour les ontologies RDF. Son principe consiste à appliquer des règles afin de découvrir des opérations de changement spécifiques et de produire des ensembles de transformation entre les versions d'une ontologie (Klein et al., 2002a) ;
- Deux extensions de l'outil PROMPTdiff – un plug-in développé pour Protégé pour la recherche de mappings entre frames en se basant sur des heuristiques (Noy & Musen, 2002) – ont été proposées dans (Klein, 2004). Leur rôle est de définir les relations d'évolution entre les éléments de deux versions d'une ontologie. l'interface utilisateur permet de visualiser certains changements complexes entre versions d'ontologie.

Un système d'évolution d'ontologie plus complet a été décrit dans (Noy et al., 2006). Le noyau du système est basé sur l'*ontologie des changements et annotations* CHAO (Change and Annotation Ontology). Les instances de l'ontologie CHAO représentent les changements entre deux versions d'une ontologie et les annotations utilisateurs liées à ces changements. Le système est implémenté sous forme de deux plug-ins de Protégé :

- Plug-in de gestion de changement donnant l'accès à une liste de changements et permettant aux utilisateurs d'ajouter des annotations de changement individuelles ou groupées et de consulter l'historique des concepts ;
- Plug-in PROMPT fournissant des comparaisons entre deux versions d'une ontologie ainsi que des informations sur les utilisateurs ayant appliqués les changements, et facilitant l'acceptation ou le rejet des changements (Noy & Musen, 2003).

IV.4- Outils de détection et de sauvegarde de changements

Dans (Plessers et al., 2007), deux prototypes d'extension de Protégé ont été présentés (pour l'approche voir section III.6) :

- Le plug-in générateur d'historique de versions créant automatiquement un historique de versions en se basant sur la trace des changements appliqués. Ces derniers sont détectés comme des événements produits dans Protégé ; l'historique des versions est alors mis à jour en donnant des indications temporelles sur les dernières versions des concepts impliqués dans les changements, et en créant les nouvelles versions appropriées de concepts représentant les nouveaux états des concepts modifiés ;
- Le Plug-in détecteur de changement prenant comme inputs un ensemble de définitions de changement et un historique de versions, et produisant en output un historique d'évolution et ce, en évaluant les définitions de changement par rapport à l'historique de versions données en entrée.

IV.5- Outil d'apprentissage et d'identification de changement dirigée par les données Text2Onto[9]

Text2Onto est un système d'apprentissage d'ontologie pour un processus automatique ou semi-automatique de création d'ontologie présenté dans (Cimiano & Völker, 2005). Text2Onto a été développé comme successeur de TextToOnto (Mäedche & Volz, 2001). Il supporte une fonctionnalité d'identification de besoins de changement dirigée par les données et inclut trois principaux composants (voir section III.1):

- GATE[10] (General Architecture for Text Engineering) comme outil de traitement de langage naturel ;
- POM comme modèle d'ontologies possibles (Model of Possible Ontologies) sauvegardant les différents changements générés et leurs explications (voir section III.1.1) ;
- Un composant de gestion de changement qui capture l'impact des changements et sélectionne parmi les possibilités de changement générées celles qui correspondent le plus au corpus considéré. La gestion de l'impact des changements est dirigée par un processus incrémental d'apprentissage (voir section III.1.1).

IV.6- Editeur ECCO[11] et outil CoSWEM[12]

Dans cette section, nous présentons l'éditeur d'ontologie ECCO (Editeur Collaboratif et Contextuel d'Ontologie) sur lequel se base le système de gestion d'évolution de web sémantique CoSWEM (Corporate Semantic Web Evolution Management) (Luong & Dieng-Kuntz, 2007).

ECCO est un éditeur offrant un environnement collaboratif et contextuel de construction d'ontologies. Il couvre les différentes étapes du cycle de développement d'une ontologie en partant de la récupération de termes extraits par des outils de traitements de texte (NLP) pour créer un vocabulaire qui sera ensuite enrichi, hiérarchisé, puis formalisé en une ontologie OWL Lite.

Il garde l'historique du processus d'élaboration d'une ontologie sous forme de métadonnées stockées au format RDF et permet aussi de générer les traces des modifications apportées à l'ontologie. Ces traces sont récupérées par le système CoSWEM pour propager les changements de l'ontologie aux annotations qui la référencent.

L'outil CoSWEM implémente l'approche décrite dans (Luong, 2007). Il applique des règles pour détecter et corriger les annotations devenues obsolètes après l'évolution d'une ontologie et se basent sur les stratégies de résolution définies dans l'approche.

V- SYNTHESE

Dans cette section, nous synthétisons l'ensemble des approches d'évolution présentées dans ce chapitre à travers un tableau récapitulant les caractéristiques générales, les fonctionnalités d'évolution supportées et les spécificités de chacune (Voir tableau page suivante).

[9] http://ontoware.org/projects/text2onto/
[10] http://gate.ac.uk/
[11] http://www-sop.inria.fr/edelweiss/projects/ewok/publications/ecco.html
[12] http://www-sop.inria.fr/edelweiss/projects/ewok/publications/coswem.html

Table 3. Synthèse des approches et outils d'évolution d'ontologie.

		Approche d'apprentissage d'ontologie basée sur l'identification des besoins de changement (Cimiano, 2007) (Cimiano & Völker, 2005)	Approche BOEMIE (Castano, 2006) (Castano et al., 2007) (Petasis, 2007)	Approche de gestion de changement pour des ontologies distribuées (Klein, 2004) (Maedche et al., 2003) (Noy et al., 2006)	Processus global d'évolution d'ontologie KAON (Stojanovic, 2004)	Approche d'évolution d'ontologie basée sur les principes de croyances (Flouris, 2006) (Flouris & Plexousakis, 2006) (Flouris et al., 2006a)	Approche de détection de changement basée sur l'historique des versions (Plessers & De Troyer, 2005) (Plessers et al., 2007)	Approche de gestion de l'évolution d'un web sémantique d'entreprise (Luong, 2007) (Luong & Dieng-Kuntz, 2007)
Caractéristiques générales	Processus d'évolution	Certaines phases	Certaines phases	Un processus de création de spécifications de changement	Processus global	Un algorithme qui prend en entrée un ensemble d'axiomes (ontologie + changement) et les applique	- Evolution à la demande - Evolution en réponse	Certaines phases
	Langage d'ontologie	OWL DL	OWL	OWL	KAON	Certaines DL en monde ouvert mais pas OWL	OWL DL	RDF(S)
	Outil / Prototype	Text2Onto	Ontology Evolution Toolkit , un composant du prototype BOEMIE	- OntoView - PROMPTdiff - Plug-ins de Protégé	Suite d'outils KAON		Deux Plug-ins de Protégé : - Générateur d'historique de versions - Détecteur de changements	Outil CoSWEM (basé sur l'éditeur ECCO)
Fonctionnalités	Identification des besoins de changement	- Dirigée par les données - Dirigée par les utilisateurs	A partir de raisonnement sur des sources multimédia et de sources de connaissances externes (ontologies ou taxonomies existantes, etc.)		- Dirigée par les données - Dirigée par l'usage	Par identification de nouveaux faits (opérations de changement centrées fait)		
	Spécification des changements	Modèle d'apprentissage	ABox (instances de concepts et de	- Ontologie d'opérations de	-Spécification des changements	Axiomes basés sur la théorie AGM	Langage de définition de	- Une ontologie d'évolution

			d'ontologie LOM	relations)	changement et de transformations entre versions (méta-modèle OWL étendu) - langage de spécification de changement (en RDF) - Ontologie des changements et annotations CHAO	KAON - Ontologie d'évolution		changement CDL (définition déclarative et formelle basée sur la logique temporelle)	inspirée de (Stojanovic, 2004) - Classification selon l'impact sur les annotations (correction obligatoire ou facultative)
	Maintenance de la cohérence	Niveau	Logique	Structure et Logique		- Structurel (KAON) - Structurel et logique (OWL Lite) (Haase & Stojanovic, 2005)	Logique	Logique	- Structure de l'ontologie - Cohérence des annotations
		Vérification	A partir des degrés de confidence	Oui (pour le peuplement et l'enrichissement)	Compatibilité entre différentes versions	- A *priori* (KAON) - Localisation de la sous-ontologie incohérente minimale après des opérations d'ajout (OWL Lite) (Haase & Stojanovic, 2005)	Révision de la satisfaction des axiomes	Algorithme de localisation	- Vérification (pour les annotations) basée sur le modèle de cohérence des annotations (exprimé en RDF) ou sur la comparaison de versions d'ontologie
		Proposition de résolution	Plusieurs solutions triées par une fonction d'évaluation		Dériver des changements additionnels	- stratégies de résolution proposant les axiomes à supprimer (niveau logique, OWL Lite)		Proposition de règles pour diminuer les axiomes les plus restrictifs	- Bibliothèque de stratégies de résolution par type de changements (pour les ontologies) - Sélection (basée sur des règles) des entités cohérentes dans les annotations (si sans

									trace de changements d'ontologie)
		Résolution automatique	Suppression des axiomes incertains	En cas de peuplement : élimination des redondances et incohérences logiques	Résoudre des problèmes spécifiques	- Approche déclarative et procédurale (KAON) - Règles de réécriture (niveau structurel OWL Lite)	Basée sur les principes de changement de croyance		- Approche procédurale de résolution des annotations (si trace de changements d'ontologie)
	Propagation des changements	Cible		- Sources de données	- Sources de données - Ontologies	- Ontologies - Applications - Instances		- Ontologies - Applications	Annotations sémantiques
		Type		Définition et mise à jour de mappings des connaissances entre l'ontologie de domaine et les sources de connaissances externes concernées par l'enrichissement	-Analyse de compatibilité entre les versions d'une ontologie et les sources de données, et propagation partielle aux données - Une proposition pour la synchronisation d'ontologie	Diffusion vers les artefacts avec différentes synchronisation		Les mainteneurs des artefacts décident	- Systématique si changement avec trace - Différée si changement sans trace
	Détection des changements				- Relations conceptuelles et d'évolution entre versions - Générer quelques informations sur les changements (ontologies RDF)			Oui	Basée sur la comparaison de versions ou la récupération de traces de changement
	Versioning	Version évoluée		Génération d'une version évoluée		Sauvegarde de la version évoluée		Sauvegarde de la version évoluée	
		Comparaison des versions			Oui				Oui

		Gestion de plusieurs versions			Oui				Oui
	Log d'évolution	Changements appliqués	Oui	Oui	Oui (les changements détectés)	Oui		Sauvegarde des changements détectés : - Historique des versions de chaque entité de l'ontologie - Historique de l'évolution d'une version de l'ontologie	
		Trace des opérations de gestion	Traçabilité des explications des changements.			Traçabilité de tout le processus de gestion (journal d'évolution)			
	Spécificités		Evolution intégrée à l'apprentissage d'ontologie	Peuplement et enrichissement d'ontologie guidés par des patrons.	Un Framework de synchronisation de versions d'ontologies distribuées	Processus global et un système dédié	Application de la théorie AGM comme approche complémentaire pour avoir un formalisme formel et un processus automatique.	Plusieurs vues (interprétations) de changements selon les différents utilisateurs	Stratégies de détection et de correction des incohérences des annotations sémantiques

VI- DISCUSSION

Plusieurs observations se dégagent de l'analyse des problématiques émanant de la gestion des changements d'ontologie et l'étude des approches existantes en évolution d'ontologie :

- Pour assurer l'application d'un changement, un processus d'évolution d'ontologie doit permettre de dériver l'ensemble des changements intermédiaires nécessaires à son application, analyser l'impact des changements et maintenir la cohérence de l'ontologie évoluée et de ses artefacts dépendants ;
- Un processus d'évolution doit être automatisé et optimisé tout en restant suffisamment flexible pour permettre à l'utilisateur de valider ou révoquer l'application des changements et des résolutions ;
- Un processus d'évolution devrait offrir la possibilité d'identifier les changements utiles au raffinement et à l'amélioration de l'ontologie en tenant compte du domaine qu'elle modélise, des applications et usages de l'ontologie ainsi que de la qualité de l'ontologie elle-même.

Au-delà des résultats obtenus et des réalisations accomplies en évolution d'ontologie, il reste encore des questions ouvertes à la réflexion. Selon notre point de vue, ces questions s'appuient principalement sur les observations suivantes :

- La plupart des implémentations existantes se focalisent relativement plus sur les changements simples et élémentaires que sur les changements complexes. Certes, les compositions de changements complexes ne peuvent être définies exhaustivement ; mais la question qui se pose est : comment fournir en pratique, des lignes directrices pour guider leur application de manière plus optimisée et plus automatisée ? L'analyse de cas d'utilisation en évolution d'ontologie est nécessaire afin de définir des procédures et des directives spécifiques à la gestion de changements complexes ;
- Les dimensions collaborative et distribuée d'un environnement d'ontologie doivent être considérées plus en détail parce qu'en pratique, d'une part, les ontologies sont plutôt maintenues de manière centralisée et d'autre part, les artefacts dépendants sont souvent concernés par leur évolution. Les recherches en propagation de changements, validation globale, résolution de conflits et même en raisonnement sur des ontologies incohérentes lorsque des accords mutuels ne peuvent être trouvés, sont à approfondir ;
- Un système d'évolution d'ontologie doit être enrichi par un méta-modèle ou une sorte de couche générique indépendante du langage permettant de modéliser des directives génériques de gestion de changement pouvant être utilisées pour différents processus d'évolution. Par ailleurs, de tels méta-modèles pourraient guider également la propagation des changements aux ontologies dépendantes.

VII- CONCLUSION

Avec l'utilisation étendue des ontologies dans les applications industrielles et académiques, l'évolution d'ontologie est aujourd'hui, un axe de recherche essentiel. Dans ce chapitre, nous avons exposé les problématiques et les besoins en évolution d'ontologie, et présenté une étude comparative des évolutions d'ontologie, de schéma de bases de données et de système à bases de connaissance. Une synthèse de l'état de l'art des principales approches existantes a aussi été exposée. Nous avons également, présenté des outils supportant complètement ou en partie, le processus d'évolution d'ontologie. Avant de conclure, nous avons discuté des questions ouvertes et réflexions à approfondir en gestion de changements d'ontologie.

Dans le chapitre suivant, nous présentons notre approche d'évolution d'ontologie OWL DL que nous avons défini dans le cadre de cette thèse, en détaillant ses principes ainsi que son processus de gestion de changement et ses différentes phases.

Chapitre 3 : Onto-Evo^{a}l : une approche d'évolution d'ontologie

Résumé : L'utilisation croissante des ontologies et le coût de leur adaptation aux changements appuient la nécessité de gérer l'évolution d'ontologie de manière méthodique et formelle. Dans nos travaux, nous nous intéressons particulièrement aux problèmes inhérents à la gestion des changements d'une ontologie dans un contexte local et nous présentons dans ce chapitre une approche d'évolution d'ontologies à base de patrons. Les patrons modélisés correspondent aux dimensions changement, incohérence et alternative de résolution. Sur la base de ces patrons et des liens conceptuels entre eux, nous proposons un processus automatisé permettant de guider et contrôler l'application des changements tout en assurant la cohérence de l'ontologie évoluée. La méthodologie intègre aussi une activité d'évaluation basée sur un modèle de qualité d'ontologies. Ce modèle est employé pour guider la gestion des incohérences en évaluant l'impact des résolutions proposées sur la qualité de l'ontologie et ainsi choisir celle qui préserve la qualité de l'ontologie évoluée.
La gestion des changements étant fortement liée au modèle dans lequel est représentée l'ontologie, nous nous focalisons sur le langage OWL et nous tenons compte de l'impact des changements sur la cohérence logique de l'ontologie telle que spécifiée dans la couche OWL DL.

I- INTRODUCTION

L'utilisation croissante des ontologies notamment sur le web, rend indispensable la gestion de leur évolution. Même en adoptant une approche méthodique de conceptualisation d'ontologie, certaines erreurs ou irrégularités peuvent être décelées à l'usage (Helfin, 2001). La représentation et la manipulation des connaissances ontologiques doit tenir compte de l'évolution du domaine qu'elles modélisent, de la dynamicité de leur environnement d'application et des changements des objectifs d'usage. Et même en amont de leur usage, les connaissances ontologiques sont issues d'un cycle de développement itératif et incrémental.

Appliquer des changements à une ontologie vise normalement, à la modifier pour la rendre plus précise et plus adéquate au domaine qu'elle modélise et à ses objectifs d'usage. La modification d'ontologie présente plusieurs difficultés aussi bien sur un plan pratique que théorique. Il n'est pas toujours évident d'appréhender clairement ce qui est attendu d'un besoin de changement, ni comment un changement peut être réalisé. Le coût de l'adaptation des ontologies aux changements appuie la nécessité de gérer l'évolution d'ontologie de manière méthodique et formelle.

Dans nos travaux, nous nous intéressons aux problèmes inhérents à la gestion des changements d'une ontologie dans un contexte local (chapitre 2, figure 1). Conduire l'application des changements tout en maintenant la cohérence de l'ontologie est une tâche cruciale et couteuse en termes de temps et de complexité. Un processus automatisé est donc essentiel. Mais comment conduire l'application d'un changement tout en assurant la cohérence de l'ontologie évoluée ? Comment analyser les effets d'un changement et les résoudre ? Et si plusieurs solutions sont possibles, laquelle choisir et selon quels critères ?

Pour répondre à toutes ces questions, nous avons défini une méthodologie de gestion de changement *Onto-Evo^{a}l* (Ontology Evolution-Evaluation) qui s'appuie sur une modélisation à l'aide de patrons. Ces

patrons spécifient des classes de changements, des classes d'incohérences et des classes d'alternatives de résolution. Sur la base de ces patrons et des liens entre eux, nous proposons un processus automatisé permettant de conduire l'application des changements tout en maintenant la cohérence de l'ontologie évoluée. La méthodologie intègre aussi une activité d'évaluation basée sur un modèle de qualité d'ontologies. Ce modèle est employé pour guider la gestion des incohérences en évaluant l'impact des résolutions proposées sur la qualité de l'ontologie et ainsi choisir celle qui préserve la qualité de l'ontologie évoluée.

La gestion des changements étant fortement liée au modèle dans lequel est représentée l'ontologie, nous nous focalisons sur le langage OWL et nous tenons compte de l'impact des changements sur la cohérence logique de l'ontologie telle que spécifiée dans la couche OWL DL.

Le chapitre est organisé comme suit : dans la section 2, nous énonçons les principes de l'approche *Onto-Evo*^{a}*l*. Dans la section 3, nous décrivons son architecture générale. Le processus de gestion de changement est détaillé dans la section 4. La section 5 expose notre démarche de sauvegarde de la trace d'évolution. Avant de synthétiser les différents points de l'approche, une discussion est présentée à la section 6.

II- PRINCIPES DE L'APPROCHE ONTO-EVO^{A}L

Faire évoluer une ontologie fait intervenir des connaissances inter-reliées et complexes et nécessite une compréhension profonde des interactions existantes entre les différents composants de l'ontologie, et la définition de mécanismes spécifiant comment les connaissances peuvent être changées et comment maintenir la cohérence de ces connaissances.

Pour assurer une maintenance cohérente, un processus d'évolution doit expliciter toute demande de changement, analyser son impact et dériver les changements intermédiaires à appliquer afin de préserver la cohérence de l'ontologie. Cette tâche est suffisamment cruciale pour ne pas être laissée à l'utilisateur, sa gestion manuelle étant très couteuse en temps et en risque d'erreurs. Il n'est en effet, pas raisonnable de s'attendre à ce qu'une intervention humaine puisse cerner une base de connaissances dans sa globalité (les connaissances implicites, les règles et contraintes, etc.) ainsi que ses interdépendances (Tallis & Gil, 1999).

L'intention principale de notre projet de recherche qui nous a menés à la définition de l'approche *Onto-Evo*^{a}*l*, est justement d'apporter des solutions explicites de guidage et d'optimisation à la problématique de gestion des changements à travers une approche méthodologique et formelle d'évolution d'ontologie.

Les principes constituant les fondements de l'approche *Onto-Evo*^{a}*l* sont : (1) le modèle d'ontologie OWL DL, (2) une modélisation par patrons de la gestion de changement, et (3) l'emploi de l'évaluation de qualité pour guider la résolution des changements. Ils sont détaillés dans les sections ci-après.

II.1- Modèle d'ontologie OWL DL

Rappelons les besoins en évolution d'ontologie présentés dans le chapitre précédant (chapitre 2, section II.1) : les besoins fonctionnels d'un processus d'évolution d'ontologie consistent à prendre en charge l'application des changements d'ontologie et à assurer la cohérence de l'ontologie évoluée. Ceci engage deux aspects cruciaux : l'analyse et la résolution des effets de changement.

L'expressivité et la sémantique du langage de représentation d'ontologie influencent les choix qui peuvent être pris quant aux détails de fonctionnement d'un processus automatisé de gestion de changement. Il nous a donc été nécessaire de choisir le modèle d'ontologie sur lequel reposera le fondement de notre approche *Onto-Evo*^{a}*l*.

Dans tous les langages d'ontologies, nous retrouvons les caractéristiques ontologiques de base à savoir, les concepts, les propriétés, les instances et l'héritage de concepts. Cependant, outre le nombre de primitives conceptuelles, ces langages diffèrent au niveau de la sémantique de leur modèle ce qui rend la définition même de la cohérence différente d'un modèle à un autre. La définition des méthodes de résolution d'incohérences à appliquer pour la validation de changements dépend de la sémantique formelle du modèle de l'ontologie.

Deux grandes classes de modèles peuvent être distinguées (Van Harmelen & Horrocks, 2000) : les modèles de *logique de description DL* et les modèles à base de *Frames*. Les DL décrivent les connaissances d'un domaine à travers des concepts atomiques correspondant à des prédicats unaires, des rôles atomiques correspondant à des prédicats binaires qui décrivent les relations entre les concepts et des individus correspondant aux objets du domaine. La maintenance de la cohérence d'un modèle DL est fondamentalement différente puisque les primitives conceptuelles sont interprétées à travers des règles d'inférence et des axiomes alors que dans les modèles basés sur des Frames, l'interprétation des primitives est contrôlée par des contraintes.

II.1.1- Logiques de description DL

Aujourd'hui, les logiques de descriptions représentent le modèle fondamental du Web sémantique, particulièrement dans la conception d'ontologies. Les formalismes de représentation de connaissances en DL se distinguent par leur sémantique formelle basée sur la logique et par l'apport de services d'inférence offrant des procédures de décidabilité complète (Sotnykova et al., 2005). Les Systèmes basés sur ces formalismes fournissent en effet, des possibilités de raisonnement variées qui déduisent la connaissance implicite de la connaissance représentée explicitement (Nardi & Brachman, 2002).

Les DL sont issues des réseaux sémantiques et du langage KL-ONE (Brachman & Schmolze, 1985). Leur sémantique formelle offre des fragments décidables de la logique de premier ordre plus structurés que celle-ci[13], et permet d'accéder aux connaissances qu'elles contiennent et de raisonner sur ces connaissances de manière formelle. Elle permet par ailleurs, d'implémenter des raisonneurs optimisés tels que Racer[14] et FaCT++[15].

Les DL décrivent le domaine en termes de concepts, de rôles et d'individus (Baader et al., 2003). Les concepts modélisent des classes d'individus. Ils peuvent être primitifs ou définis. Les rôles modélisent des relations entre classes. La relation de subsomption permet d'organiser les concepts et les rôles en hiérarchies. Les concepts et les rôles possèdent des descriptions structurelles, élaborées à partir d'un certain nombre de constructeurs. Une sémantique est associée à chaque description de concept et de rôle par l'intermédiaire d'une interprétation (des règles décrivant comment interpréter les constructeurs). Les manipulations opérées sur les concepts et les rôles sont réalisées en accord avec cette sémantique. Les logiques de description permettent aussi d'exprimer des relations entre concepts telles que l'inclusion ou l'équivalence et de construire des concepts nouveaux en appliquant des constructeurs de concepts (intersection, union, complément, quantification de rôles, restrictions numériques sur les rôles).

OWL est un modèle d'ontologie qui se base – dans sa couche DL – sur les logiques de description $\mathcal{SHOIN}(\mathcal{D})$ (Horrocks & Sattler, 2005). OWL-DL inclut toute la sémantique formelle des DL et les capacités de raisonnement qui en découlent. Dans notre approche, nous nous intéressons à l'évolution des ontologies OWL DL.

[13] http://www.inf.unibz.it/~franconi/dl/course/
[14] http://www.racer-systems.com/
[15] http://owl.man.ac.uk/factplusplus/

II.1.2- OWL et les logiques de description

Le langage OWL a été défini comme un standard pour la représentation d'ontologie pour le Web (recommandation du W3C, 2004)[16]. L'objectif de la définition de OWL est de fournir un langage qui facilite la manipulation et la gestion des ontologies aussi bien à un niveau syntaxique que sémantique tout en offrant une expressivité formelle supportant des mécanismes de raisonnement à différents niveaux de complexité (Horrocks, et al., 2003).

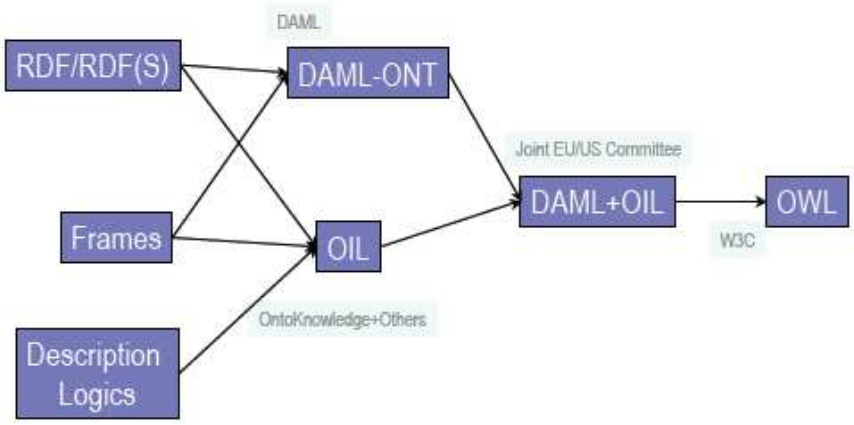


Figure 1. Arborescence de la famille du langage OWL[17].

Résultant d'une extension de standards existants, OWL se base sur des idiomes familiers de représentation de connaissances (figure 1) : il se base principalement sur les spécifications du langage DAML+OIL[18] mais reprend également les bases de RDF et XML et les améliore pour une sémantique plus formelle. Les logiques de description sont aussi au cœur du modèle OWL.

Le modèle OWL offre :

- Trois variantes[19] : OWL Full, OWL DL et OWL Lite ;
- Une stratification syntaxique ainsi qu'une stratification sémantique des constructeurs de ces variantes (OWL Full> OWL DL> OWL Lite) ;
- Et une sémantique DL « définitive » basée sur les preuves (si une partie de OWL Full contredit celle de OWL-DL alors, elle est rejetée).

De par leurs constructeurs, les variantes de OWL diffèrent par les contraintes appliquées à leurs constructeurs et par le niveau de décidabilité logique qu'elles offrent.

[16] http://www.w3.org/TR/owl-ref/
[17] Figure extraite du cours de Sean Bechhofer dans le cadre de l'école d'été SSSWO 2007.
[18] http://www.daml.org/2001/03/daml+oil-index.html
[19] http://www.w3.org/TR/owl-features/

	OWL Full	OWL DL	OWL Lite
Syntaxe	- Union de la syntaxe de OWL et de RDF.	- Des fragments de logique de premier ordre (le ¼ de DAML+OIL).	- Un sous-ensemble simpliste de OWL-DL.
Propriétés	- Aucune restriction sur le vocabulaire de OWL ; - Basé sur le modèle théorique de RDF ; - Les inférences sont gérées par des raisonneurs de logique de premier ordre ; - Les sémantiques ne remettent pas en cause celles de OWL-DL.	- Restreint l'utilisation du vocabulaire OWL (ex. un élément ne peut être à la fois classe et instance) ; - Défini par une syntaxe abstraite et un mapping à RDF ; - Basé sur le modèle théorique standard des DL et des logiques de premier ordre avec une sémantique « définitive » ; - Bénéficie des acquis des DL (une sémantique bien définie, des propriétés formelles compréhensibles en termes de complexité et de décidabilité, des algorithmes d'inférence connus et des raisonneurs DL existants).	- Un sous-ensemble des constructeurs de OWL-DL (pas de négation ou d'union explicites, cardinalité restreinte à 0 ou1, pas de nominaux OneOf) ; - Sémantique basée sur les DL, raisonnement à travers les moteurs standards de DL ; - Pas très utilisé en pratique.
Expressivité	- Expressivité complète mais décidabilité non garantie.	- Expressivité incomplète mais décidable.	- Expressivité minimale facilitant la traçabilité des inférences.

Table 1. Présentation synthétisée des trois couches de OWL.

En OWL-DL, les classes (concepts) et les propriétés (rôles) sont organisées en hiérarchie. La représentation et la manipulation des classes et des propriétés relèvent du niveau terminologique : la *TBox*. La description et la manipulation des individus relèvent du niveau factuel : la *ABox*. Les opérations qui sont à la base du raisonnement terminologique sont la classification et l'instanciation. La classification s'applique aux classes et aux propriétés et permet de déterminer la position d'une classe ou d'une propriété dans leur hiérarchie respective. La classification permet aussi d'assister la construction et l'évolution de ces hiérarchies. L'instanciation permet de retrouver les classes dont un individu est susceptible d'être une instance.

L'interprétation de la cohérence étant dépendante du modèle de représentation, nous pouvons à présent, définir la cohérence du modèle OWL DL qui est au cœur des principes de l'approche *Onto-Evo[a]l.*

II.1.3- Cohérence du modèle OWL DL

A ce niveau du chapitre, nous discutons des deux niveaux de cohérence : *cohérence structurelle* et *cohérence logique*. Nous considérons ainsi, une ontologie cohérente comme étant une ontologie dont les entités sont en accord avec la spécification formelle du modèle OWL DL et dont les assertions ne présentent pas de contradictions.

II.1.3.1- Cohérence structurelle

La cohérence structurelle correspond aux invariables applicables à toute ontologie OWL DL. C'est-à-dire des conditions structurelles propres aux constructeurs de OWL-DL, et des contraintes sur les axiomes et les combinaisons d'axiomes (Horrocks, et al., 2003).

II.1.3.2- Cohérence logique

La cohérence logique est directement liée à la sémantique formelle du modèle OWL DL. La signification que nous adoptons pour « cohérence logique » correspond à celle de « *logical consistency* » en anglais c'est-à-dire, la satisfaction des axiomes de l'ontologie au niveau logique.

OWL DL est un langage centré axiome. Les classes et les propriétés possèdent des descriptions structurelles, élaborées à partir d'un certain nombre de constructeurs. Une sémantique est associée à chaque description par l'intermédiaire d'une interprétation du domaine (Horrocks & Patel-Schneider, 2004). La satisfaction de l'ontologie par une interprétation est contrainte par la satisfaction de tous les axiomes de l'ontologie (axiomes de concepts, de propriétés et d'individus).

D'autres types de cohérences sont aussi pris en compte par l'approche *Onto-Evo*^{a}*l* (cohérence conceptuelle et cohérence de modélisation de domaine), ils seront discutés plus loin dans ce chapitre.

II.2- Modélisation par patrons

Comme il a été discuté dans le chapitre état de l'art (chapitre 2), les besoins de contrôler et automatiser le processus d'évolution d'ontologie sont conséquents. Les éditeurs d'ontologie existants n'offrent pas de réelles fonctionnalités d'évolution (analyse et résolution des changements, maintenance de la cohérence, etc.). Pourtant, la gestion des changements d'ontologie est une tâche complexe. Les utilisateurs d'ontologie qu'ils soient des ingénieurs système ayant besoin de manipuler des ontologies sans pour autant devenir des spécialistes, ou des ontologues « novices » ou même des ingénieurs d'ontologie, ne peuvent appréhender tous les effets de changement, évaluer leur impact sur l'ontologie et les résoudre.

L'approche *Onto-Evo*^{a}*l* apporte une réponse à ces besoins à travers la modélisation par patrons des pratiques de gestion de changement sous forme d'une librairie partagée de patrons de gestion de changement *CMP* (Change Management Patterns). Les patrons CMP sont introduits comme un moyen de faciliter l'application d'un changement tout en assurant la cohérence de l'ontologie évoluée. Ils modélisent des structures récurrentes de changements, des incohérences logiques qu'ils peuvent potentiellement causer et des alternatives pouvant résoudre ces incohérences constituant ainsi, un potentiel prometteur pour le guidage du processus d'évolution.

II.3- Evaluation de la qualité pour guider la résolution de changement

Afin d'aller plus loin dans l'automatisation et l'optimisation, nous avons introduit dans le processus d'évolution, des techniques d'évaluation de qualité que nous avons employées pour évaluer l'impact des solutions de résolution générées par la gestion des changements, sur la qualité de l'ontologie évoluée. L'idée étant de classer les résolutions proposées – quand plusieurs solutions sont possibles – pour n'en choisir que celle qui préserve – si ce n'est améliore – la qualité de l'ontologie et ainsi, guider le choix des résolutions à appliquer plutôt que de solliciter l'intervention de l'ingénieur d'ontologie directement après la génération des résolutions d'incohérence, comme le font la plupart des approches existantes d'évolution d'ontologie (chapitre 2).

III- ARCHITECTURE GENERALE DE L'APPROCHE ONTO-EVO[A]L

Afin de conduire de manière automatisée et optimisée l'application d'un changement tout en assurant la cohérence de l'ontologie, il est nécessaire de spécifier explicitement le changement requis, d'analyser les incohérences dues à son application, de proposer des solutions pour les résoudre, de guider l'ingénieur dans le choix des résolutions et de l'assister dans la validation finale (figure 1, chapitre 2).

Pour réaliser ces objectifs, nous avons défini des patrons de gestion de changement *CMP* (Change Management Patterns) guidant le processus d'évolution de l'approche *Onto-Evo[a]l* au niveau des trois phases clés : la spécification du changement, l'analyse du changement et la résolution du changement (figure 2). Trois catégories de patrons sont modélisées : des patrons de changements (Change Patterns), des patrons d'incohérences (*Inconsistency Patterns*) et des patrons d'alternatives (Alternative Patterns). Ces patrons permettent de définir et de classer – respectivement – les types des changements gérés par le processus en se basant sur le modèle OWL DL, les types des incohérences logiques supportées par le processus en se référant aux contraintes logiques de OWL DL et les types des alternatives de résolution d'incohérence pouvant être générées.

Toujours dans le but de guider le processus d'évolution de manière automatisée et optimisée, nous avons également défini un module d'évaluation de qualité permettant de choisir les résolutions les plus pertinentes. L'évaluation se base sur un modèle de qualité évaluant l'impact des résolutions générées par le processus, sur le contenu et l'usage de l'ontologie à travers un ensemble de métriques quantitatives et ce, afin de choisir une résolution qui préserve la qualité de l'ontologie.

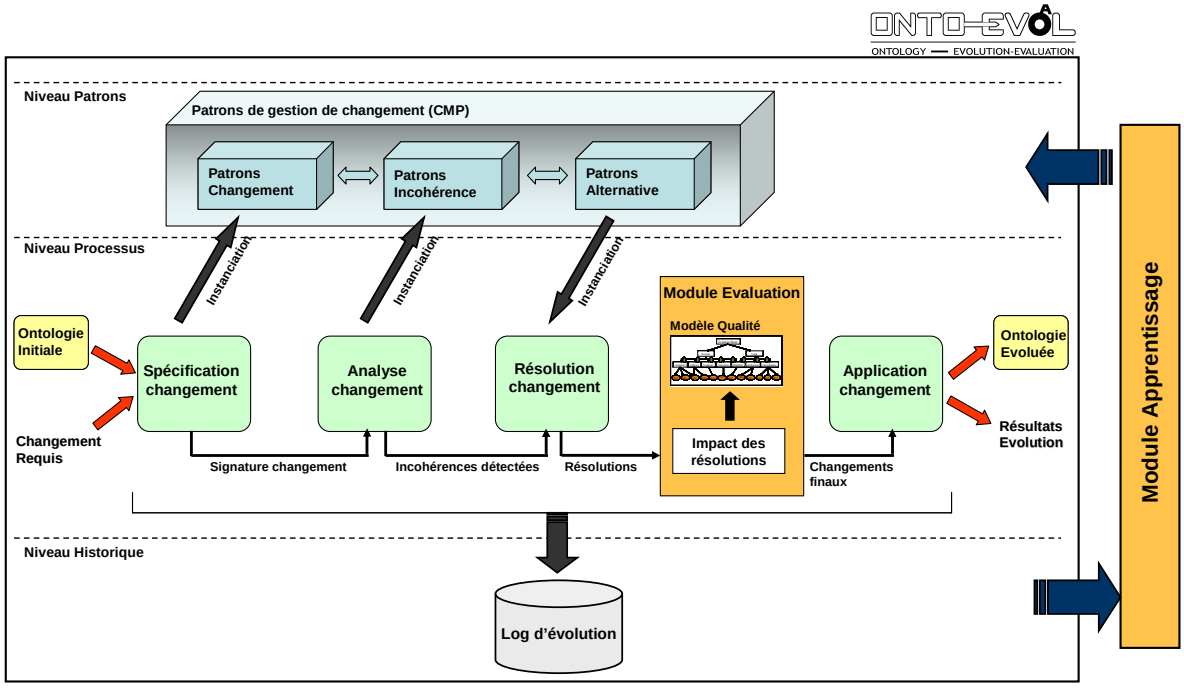


Figure 2. Architecture générale de l'approche *Onto-Evo[a]l*.

Ainsi, l'architecture générale de l'approche (figure 2) *Onto-Evo[a]l* est structurée selon trois niveaux :

- Le niveau *processus* qui représente le cœur de l'approche. Il comprend les composants fonctionnels de l'approche à savoir : les quatre phases principales de gestion de changement et le module d'évaluation de qualité guidant le choix des résolutions. Le fonctionnement de ces composants sera détaillé dans la section suivante (section IV) ;

- Le niveau *patrons* qui représente le méta-modèle guidant le déroulement du processus. Il est décrit par les patrons *CMP* (patrons de changements, patrons d'incohérences et patrons d'alternatives) et les liens conceptuels entre eux. La modélisation des patrons *CMP* par une ontologie OWL DL l'*ontologie CMP* spécifiant les différentes classes de changements, d'incohérences et d'alternatives gérées par l'approche *Onto-Evo*^{a}*l* et les relations sémantiques entre elles, est détaillée dans le chapitre suivant (chapitre 4) ;
- Le niveau *historique* qui assure la traçabilité des changements et des traitements effectués le long du processus à travers le log d'évolution sauvegardant tout l'historique. La gestion de la traçabilité sera développée ultérieurement (section V).

Outre ces trois niveaux, l'architecture de l'approche prévoit un module d'apprentissage permettant d'employer l'historique d'évolution pour l'enrichissement du méta-modèle guidant le processus.

Le point de lancement du processus est la phase de spécification de changement permettant de décrire formellement la sémantique du changement à appliquer sur l'ontologie (le changement requis). La phase de spécification est guidée par l'instanciation du patron de changements correspondant au changement requis (figure 2). La phase suivante a pour objectif d'analyser l'impact du changement en explicitant et localisant les incohérences causées. Elle se base sur la classification des incohérences détectées en fonction des patrons d'incohérences qui leur correspondent. A la phase de résolution de changement, l'instanciation des patrons se fait dans le sens inverse que précédemment, c'est-à-dire, du niveau patrons vers le niveau processus en générant les instances d'alternatives de résolution à partir des liens conceptuels définis entre les patrons d'incohérences – instanciés dans la phase précédente – et les patrons d'alternatives les résolvant (figure 2). Après, les alternatives sont combinées pour constituer les résolutions du changement. Les résolutions proposées sont alors évaluées et triées par le module d'évaluation selon leur impact sur la qualité de l'ontologie, ce qui permet de sélectionner celle qui préserve la qualité et de l'appliquer conjointement au changement requis pour faire évoluer l'ontologie (figure 2). Tous les résultats du processus sont enregistrés dans le log d'évolution afin de sauvegarder l'historique de gestion des changements de l'ontologie (figure 2).

IV- PROCESSUS DE GESTION DE CHANGEMENT

Le processus défini par l'approche *Onto-Evo*^{a}*l*, permet de déterminer à la demande d'un changement, ce qui doit être modifié et comment le changer ; et conduire l'application du changement requis tout en assurant la cohérence de l'ontologie. Il permet également de tenir compte de l'impact sur la qualité de l'ontologie pour justifier les choix de résolution ce qui optimise la gestion des changements et facilite la compréhension de l'évolution de l'ontologie.

Le processus est conduit à travers quatre phases (figure 3) : spécification du changement, analyse du changement, résolution du changement et application du changement.

IV.1- Spécification du changement

La phase de spécification est lancée suite à la demande d'un changement à appliquer sur une ontologie initiale supposée cohérente (figure 3 **(1)**). L'utilisateur demande un changement basique ou composé (section III.3.3, chapitre 2) sans avoir à définir et ordonner les opérations de changements intermédiaires nécessaires à son application.

La phase de spécification a un rôle plus large qu'une simple représentation du changement requis dans le langage de l'ontologie. Elle vise à expliciter le changement de manière formelle et compréhensible pour

préparer les phases d'analyse et de résolution. Le changement est tout d'abord explicité et formalisé dans le modèle OWL DL (figure 3 **(2)**). Ensuite, il est classé selon les types prédéfinis par les patrons de changements et enrichi par un ensemble d'arguments permettant de préparer les phases suivantes (figure 3 **(3)**). Le patron correspondant est instancié aux spécificités réelles du changement requis (références des entités concernées, les changements intermédiaires qui le composent, les valeurs de changement, etc.). La spécification du changement sera explicitée à la description des patrons de changements dans le chapitre 4. L'ensemble des spécifications dérivées constitue la signature du changement.

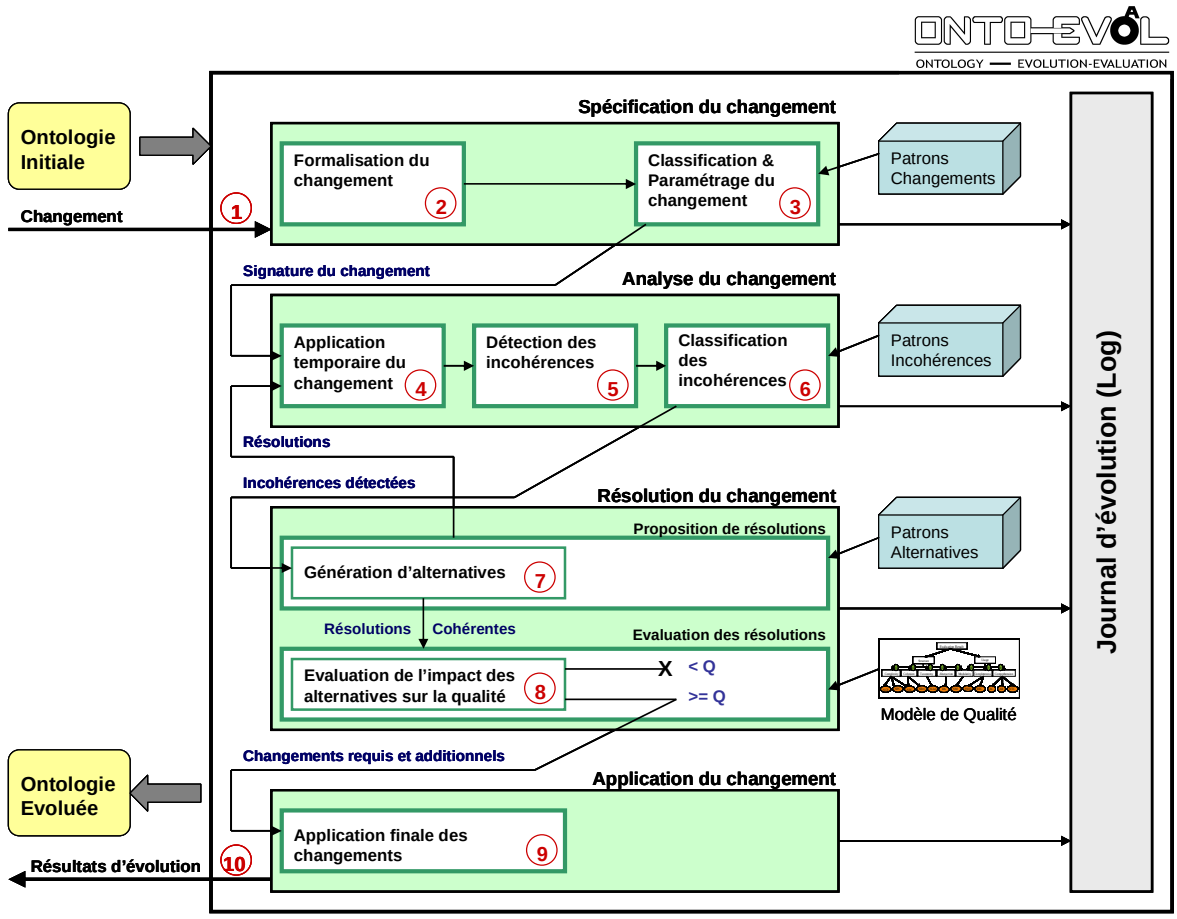


Figure 3. Processus d'évolution de l'approche *Onto-Evo[a]l*.

IV.2- Analyse du changement

Au cours de la phase d'analyse du changement, le changement formellement spécifié est appliqué à une version temporaire de l'ontologie pour analyser ses impacts (figure 3 **(4)**). L'étape suivante permet de détecter les incohérences causées (figure 3 **(5)**). Les incohérences détectées sont alors classées selon les types prédéfinis par les patrons d'incohérences (figure 3 **(6)**). L'instanciation des patrons d'incohérences permet de préparer la phase de résolution notamment à travers le lien avec les patrons d'alternatives (Chapitre 4).

IV.2.1- Niveaux de cohérence

L'analyse du changement tient compte des deux niveaux de cohérence : la cohérence structurelle et la cohérence logique.

La cohérence structurelle se référant aux contraintes syntaxiques du langage OWL DL et à l'utilisation de ses constructeurs, est assurée à travers la spécification formelle des changements à appliquer par les patrons de changements dont la modélisation s'est basée sur les fondements du modèle OWL DL.

L'analyse de la cohérence logique permet de vérifier que l'ontologie est sémantiquement correcte et ne comporte pas de contradictions.

IV.2.2- Cohérence logique

L'application d'un changement – même basique – peut avoir un impact sur la cohérence logique de l'ontologie : la suppression d'une classe par exemple, nécessite de prendre une décision concernant les propriétés de cette classe, ses sous-classes et ses instances (les supprimer, les reclasser, etc.). L'impact est encore plus conséquent lorsqu'il s'agit d'un changement composé. Même si les effets de chaque changement intermédiaire – pris à lui seul – sont mineurs, l'effet cumulatif de tous les changements qui le composent, peut être considérable.

OWL DL est construit sous l'hypothèse du monde ouvert, le contenu de l'ontologie ne représente qu'un sous-ensemble des connaissances du monde réel qu'elle modélise. L'hypothèse du monde ouvert facilite le raisonnement sur les définitions intensionnelles des classes (l'ensemble de ses propriétés), l'inférence d'instances et la déduction de négation (si un individu *i* n'est pas une instance de la classe *c* alors, on peut déduire qu'il doit être individu de $\neg c$). L'hypothèse du monde ouvert stipule également que le peuplement d'ontologie doit être souple et ne doit pas exiger des instanciations complètes respectant les cardinalités. Cependant, une opération de changement liée au peuplement de l'ontologie telle que l'instanciation d'une classe par un individu qui existe dans l'ontologie et qui est déjà affecté à d'autres classes ou la modification d'une relation d'instanciation, peut altérer la cohérence logique d'une ontologie. La spécification du changement à elle seule, ne suffit pas pour déterminer l'impact sur la cohérence logique de l'ontologie, il faut vérifier tous les axiomes se référant à l'instance ajoutée/modifiée.

Il n'est pas évident de déterminer intuitivement l'impact d'un changement sur la cohérence logique de l'ontologie. L'objectif de la modélisation des patrons CMP vise justement, à apporter un élément de réponse à ce problème et à guider la phase d'analyse en définissant – dans les propriétés des patrons de changements – un ensemble de contraintes à vérifier à l'application du changement modélisé ainsi qu'un lien conceptuel entre les patrons de changements et les patrons d'incohérences permettant de prévoir les éventuels types d'incohérences qui peuvent potentiellement être causés par un type de changement. La modélisation des CMP sera détaillée dans le chapitre 4.

Ces incohérences potentielles prévues par les patrons aident à orienter la vérification de la cohérence de l'ontologie. De plus, une analyse concrète des incohérences réellement causées est effectuée. Elle se base sur l'emploi du raisonneur Pellet (Sirin et al., 2007) en interfaçage avec le système de gestion de changement (chapitres 5 et 7).

L'objectif de la phase d'analyse de changement est de vérifier la cohérence logique et de délimiter la partie incohérente de l'ontologie. La localisation des incohérences s'inspire de l'algorithme de localisation présenté dans (Haase & Stojanovic, 2005), permettant de déterminer la sous-ontologie *O'* incohérente minimale telle que $O' \subseteq O$ (*O* étant l'ontologie analysée) et $\forall O'' \subset O'$, *O''* est une sous-ontologie cohérente de *O*. Cette partie sera détaillée dans le chapitre 5.

IV.3- Résolution du changement

La résolution du changement comprend deux activités principales : proposition de résolutions et évaluation des résolutions.

IV.3.1- Proposition de résolutions

A partir des instances d'incohérences détectées, les patrons d'alternatives sont instanciés pour générer les alternatives potentielles de résolution (figure 3 **(7)**). Chaque alternative représente des opérations de changement à appliquer pour résoudre une incohérence. La combinaison des différentes alternatives proposées pour chaque incohérence, permet de constituer les résolutions du changement. Cependant, ces résolutions ne doivent pas causer des incohérences. C'est pourquoi, toutes les résolutions sont vérifiées selon un mécanisme récursif et seules les résolutions cohérentes sont retenues. Les patrons d'alternatives de résolution seront détaillés dans le chapitre 4.

Tenant compte du principe de « continuité ontologique » qui stipule qu'une connaissance existante ne peut être infirmée (Xuan et al., 2006), nous tendons à minimiser les alternatives de suppression d'axiomes en proposant des alternatives de division de classes, de fusion de classes, de généralisation, de redistribution d'instances, etc. Cependant, si une résolution sous forme de suppression est nécessaire, elle est appliquée tout en vérifiant son impact sur l'ontologie.

Prenons un exemple simple illustrant la résolution d'une incohérence de disjonction causée par un changement d'instanciation :

Si *i* une instance la classe C_1, sous-classe de la classe *C*, est assignée à la classe C_2, également sous-classe de *C* et disjointe de C_1, au lieu de supprimer l'axiome de disjonction ou l'axiome d'instanciation de *i* à C_1, comme le propose certains travaux en évolution d'ontologie notamment dans (Haase & Stojanovic, 2005) (déterminer les axiomes à supprimer pour résoudre l'incohérence), l'incohérence de disjonction peut être résolue par les alternatives suivantes (figure 4) :

- L'instance *i* est redistribuée et assignée à la classe mère *C* plutôt qu'à ses sous-classes, ainsi la sémantique de l'instance *i* est gardée tout en tenant compte de l'opération de changement, et l'axiome de disjonction reste vrai ;
- Une nouvelle sous-classe C_{12} est ajoutée à la classe *C*, l'instance *i* lui sera assignée. La classe C_{12} représentera l'union des sous-classes C_1 et C_2, elle n'est pas concernée par l'axiome de disjonction et ce dernier n'étant pas supprimé, la sémantique de la disjonction entre C_1 et C_2 est conservée.

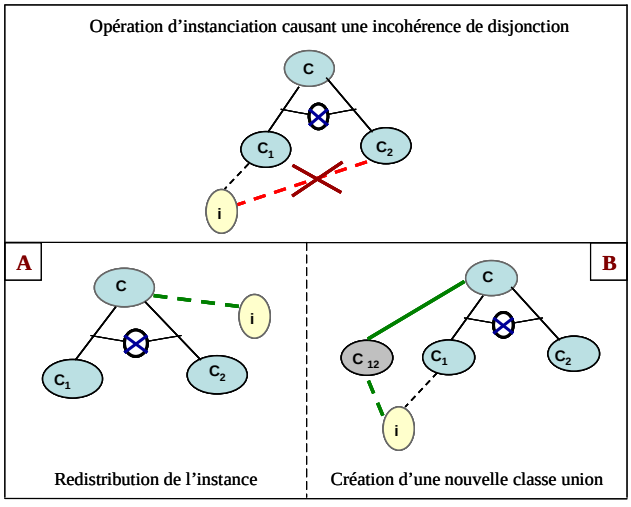


Figure 4. Les alternatives de résolution d'une incohérence de disjonction causée par une instanciation.

IV.3.2- Evaluation des résolutions

Plusieurs résolutions cohérentes peuvent être proposées pour un changement. Les résolutions doivent prendre en considération la spécificité du domaine modélisé et l'objectif d'usage de l'ontologie tout en préservant une bonne qualité de modélisation de l'ontologie.

Il y a certes, différentes façon de résoudre un changement, mais les solutions de résolution ne devraient pas découler simplement d'un raisonnement logique cherchant à éliminer des contradictions. Ceci est très réducteur de la sémantique conceptuelle de l'ontologie ainsi que de celle du changement requis. Considérer l'impact des résolutions sur la qualité de l'ontologie permet de choisir la résolution la plus adéquate ou tout ou moins, de recentrer les propositions de résolution vers un ensemble plus adapté en tenant compte de critères de qualité prédéfinis.

Ainsi, plutôt que de présenter les différentes propositions de résolution à l'ingénieur d'ontologie, nous proposons de guider le choix des résolutions à appliquer, en évaluant l'impact de chacune des résolutions sur la qualité de l'ontologie (figure 3 **(8)**). L'évaluation se base sur un modèle de qualité considérant un ensemble de critères – relatifs au contenu et à l'usage de l'ontologie – évalués à travers des métriques quantifiables.

IV.3.2.1- Niveaux de cohérence

A ce niveau du processus, nous distinguons deux autres niveaux de cohérence considérés dans l'approche *Onto-Evo*^{a}*l* :

- La *cohérence conceptuelle* correspondant à des contraintes de bonne conceptualisation. Ce niveau de cohérence, engagé déjà dans la modélisation même des patrons de changements et d'alternatives, est vérifié moyennant les critères défini dans le modèle de qualité ;
- La *cohérence de modélisation de domaine* confrontant la conceptualisation de l'ontologie au domaine qu'elle modélise à travers des critères évaluant l'usage de l'ontologie par rapport aux sources de domaine, définis dans le modèle de qualité.

IV.3.2.2- Critères d'évaluation

Les critères considérés par le modèle de qualité sont :

- La *complexité* évaluant les liens structurels et sémantiques entre les entités de l'ontologie, et la navigabilité dans la structure de l'ontologie;
- La *cohésion* tenant compte des composants connectés de l'ontologie (classes et instances) ;
- La *conceptualisation* évaluant la richesse de la conception du contenu de l'ontologie ;
- L'*abstraction* c'est-à-dire le niveau d'abstraction des classes (généralisation/spécialisation) en considérant la profondeur des hiérarchies de subsumption;
- La *complétude* considérant le degré de couverture des sources représentatives du domaine modélisé et de ses propriétés pertinentes en tenant compte de la conformité des désignations (les labels) des classes et des propriétés aux mots clés du domaine ;
- La *compréhension* estimant la facilité de compréhension de l'ontologie à travers le nommage des entités et les annotations décrivant les définitions des classes et des propriétés, et aussi la facilité d'explication des différenciations entre les classes d'une même hiérarchie (les sous-classes et la classe mère).

En reprenant l'exemple de la figure 4, l'alternative *A* n'altère pas le contenu de l'ontologie, l'alternative *B* par contre, augmente la complexité de celle-ci[20]. Relativement à l'usage, dans l'alternative *A*, l'instance *i* n'est plus rattachée à la classe C_1, si le corpus représentatif du domaine montre une importante assignation de cette instance à la classe C_1, le critère de complétude sera altéré. Le choix final des résolutions tiendra compte de ces évaluations mais aussi des pondérations attribuées par l'expert du domaine et l'ingénieur d'ontologie aux différents critères définis dans le modèle de qualité. La description détaillée du modèle de qualité et de l'activité d'évaluation sera présentée dans le chapitre 6.

IV.4- Application du changement

Cette phase correspond à l'application finale du changement. Elle est optimisée par l'emploi de techniques d'évaluation de la qualité permettant de guider la gestion des incohérences à travers des résolutions évaluées et triées, et de minimiser la dépendance à l'utilisateur. Ainsi, si une résolution préserve la qualité, le changement requis et ses changements dérivés seront directement validés et appliqués (figure 3 **(9)**) et l'ontologie évoluée. Si par contre, les résolutions ont un impact négatif sur la qualité, les résultats des différentes phases du processus seront présentés à l'ingénieur d'ontologie en complément à son expertise pour qu'il décide du changement (figure 3 **(10)**). Cette phase permet de garder conjointement à l'automatisme du processus, une certaine flexibilité permettant à l'utilisateur de contrôler et de décider du changement et de sa validation finale.

L'ensemble des traitements d'analyse et de maintenance de la cohérence est appliqué à une version temporaire de l'ontologie qui peut être abandonnée si les changements sont finalement annulés, l'ontologie initiale sera alors préservée. Dans le cas contraire, une version modifiée de l'ontologie est définie.

Une fois les changements validés, de nouveaux besoins de changement peuvent être identifiés ce qui conduit à une reprise des phases d'évolution selon un processus cyclique.

V- TRAÇABILITE DE L'EVOLUTION

Il est indispensable de garder la trace des modifications effectuées sur l'ontologie et de la gestion de leur application, pour contrôler le déroulement du processus – au cours d'un cycle – et la maintenance continue de l'ontologie et de ses versions dans le temps.

Ainsi, tout au long du processus, les résultats de gestion du changement requis ainsi que les applications validées sont sauvegardés dans le journal d'évolution. Le journal d'évolution est une structure permettant de conserver l'historique des évolutions de l'ontologie et les détails de traitement de ces évolutions sous forme de séquences chronologiques d'information. Il facilite le contrôle et le suivi de l'évolution, le retour arrière, la justification des changements, la gestion des versions et dans une perspective future l'apprentissage de nouveaux patrons de changements, d'incohérences et d'alternatives.

V.1- Conception du journal d'évolution

La représentation et l'interprétation des informations sur les changements nécessitent une spécification explicite des opérations de changement pouvant être appliquées à une ontologie ainsi que de tous les résultats – intermédiaires et finaux – de gestion de changement. La spécification des opérations de changement est explicitée par la modélisation des patrons de changements (chapitre 4). La spécification des résultats de gestion est détaillée à travers la modélisation des patrons d'incohérences (description et explication des types d'incohérences logiques supportées par le processus) et des patrons d'alternatives

[20] Cet exemple illustre un changement simple engendrant une seule incohérence. Les résolutions du changement correspondent donc aux alternatives générées.

(détaillant les résolutions pouvant être générées), la définition de liens conceptuels entre patrons de changements et patrons d'incohérences (explication des impacts éventuels des différents types de changements) et entre patrons d'incohérences et patrons d'alternatives (explication des résolutions potentielles des différents types d'incohérences), et aussi à travers la description détaillée des différents types de résultats de gestion de changement : les résultats de spécification, d'analyse, de proposition de résolutions et d'évaluation de qualité, les justifications des décisions relatives aux résultats retenus, et les explications de rejet pour ceux qui ne le sont pas.

Dans le but d'assurer la compatibilité avec le modèle des ontologies de domaine gérées par l'approche *Onto-Evo*[a]*l*, et la sémantique de l'ontologie *CMP* guidant le processus et, de disposer d'un format suffisamment explicite et formel pour faciliter ultérieurement l'apprentissage de patrons, nous avons choisi de modéliser le journal d'évolution sous forme d'une ontologie OWL DL (figure 5).

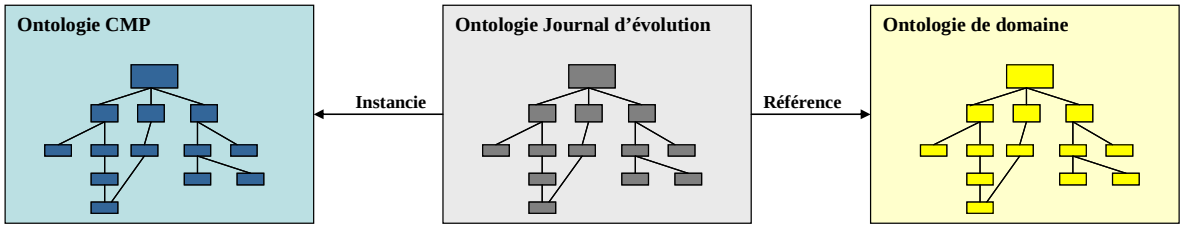


Figure 5. Liens entre l'ontologie journal d'évolution, l'ontologie CMP et l'ontologie de domaine.

V.2- Ontologie journal d'évolution

L'ontologie journal d'évolution est spécifiée par un ensemble de classes organisées en hiérarchies, de propriétés objets décrivant des relations sémantiques entre ces classes, de propriétés type de données décrivant des caractéristiques propres aux classes elles mêmes, des restrictions de valeurs sur certaines propriétés de classes et des restrictions de cardinalités.

Afin de simplifier la présentation graphique de l'ontologie journal d'évolution, nous avons choisi de la représenter (version synthétisée) en diagramme UML en nous basons notamment sur les travaux présentés dans (Brockmans et al., 2004). Ainsi, les hiérarchies de classes de l'ontologie sont représentées par des relations d'héritage entre classes, les propriétés objets par des associations entre classes en indiquant par le sens de navigabilité, le domaine et le co-domaine de la propriété, les propriétés type de données par des attributs, les restrictions de valeurs par des contraintes sur les associations correspondantes indiquant les valeurs de restrictions, et les restrictions de cardinalités par l'expression des cardinalités des associations. Par ailleurs, la propriété objet « composée de » spécifiant des compositions entre plusieurs classes de l'ontologie, est représentée par une relation d'agrégation ou de composition (agrégation forte – par valeur) de UML. La description de l'ontologie en OWL DL est présentée à l'annexe B.

L'ontologie du journal d'évolution décrite par la figure 6, modélise les trois dimensions qu'un historique d'évolution devrait prendre en compte à savoir, la dimension *trace d'exécution du processus pour la gestion d'une transaction de changement*, la dimension *trace d'évolution d'une version de l'ontologie* et la dimension *trace de versionning de l'ontologie*.

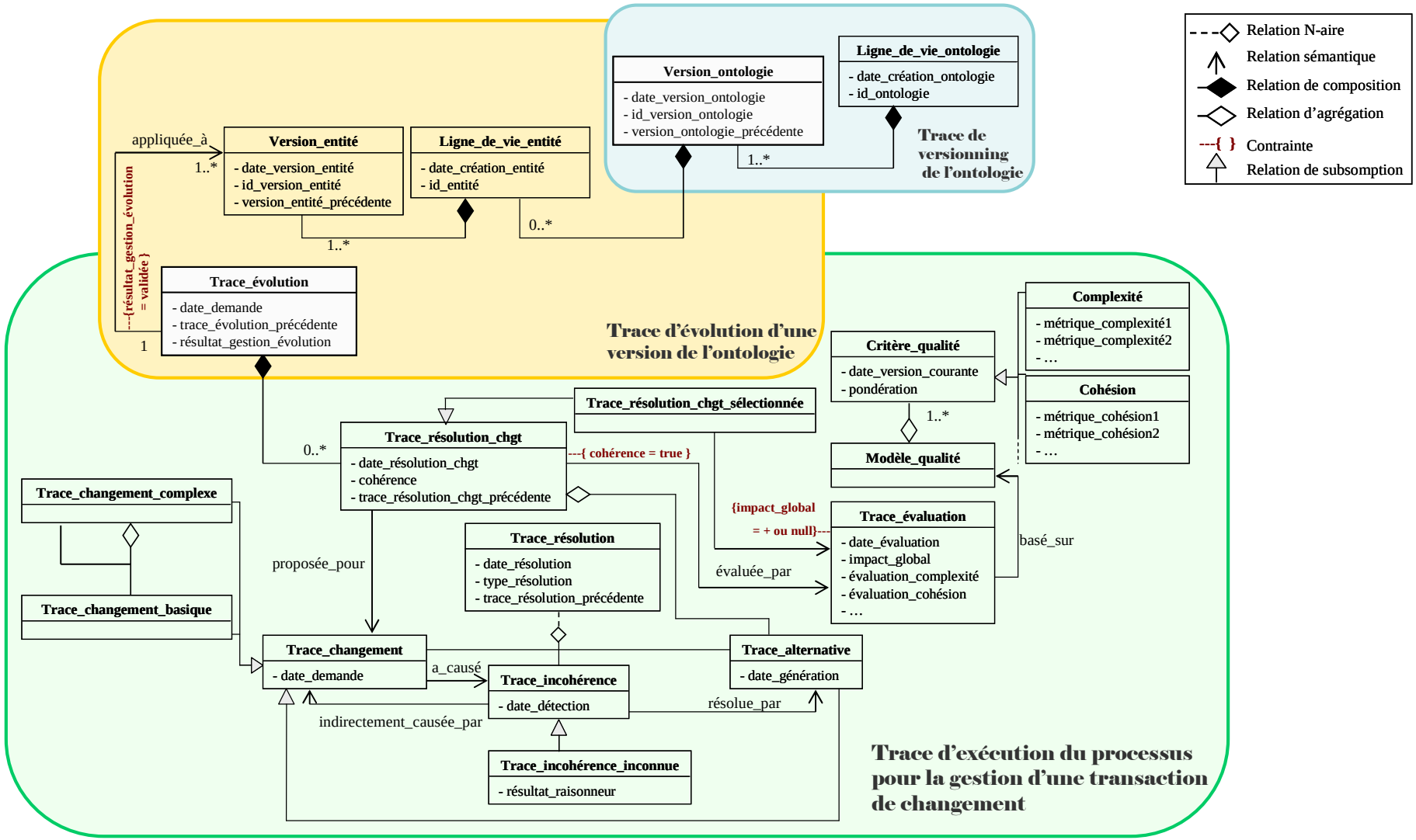


Figure 6. Représentation graphique en UML de l'ontologie journal d'évolution.

V.2.1- Trace d'exécution du processus pour la gestion d'une transaction de changement

La dimension trace d'exécution du processus pour la gestion d'une transaction de changement, est modélisée par (figure 6):

- La classe *Trace_changement* correspondant à une instance de patron de changements. Elle est décrite par les propriétés de description d'un patron changement et par la propriété *date_demande* de format timestamp (date et heure). Elle est spécialisée par les classes *Trace_changement_basique* et *Trace_changement_complexe* qui seront détaillées dans le chapitre 4.
- La classe *Trace_incohérence* correspondant à une instance de patron d'incohérences. Elle est décrite par les propriétés de description d'un patron incohérence et par la propriété *date_détection.* Elle est liée à la classe *Trace_changement* par la relation *a_causé* indiquant qu'une instance d'incohérence détectée a été causée directement par l'application du changement et la relation *indirectement_causée_par* indiquant qu'une instance d'incohérence détectée a été causée indirectement par le changement, lors du traitement des itérations de résolution.
- La classe *Trace_incohérence_inconnue* spécialise la classe *Trace_incohérence*. Elle modélise les incohérences détectées qui ne peuvent être instanciées par aucun patron d'incohérences. La propriété *résultat_raisonneur* décrit le résultat d'analyse de cette incohérence par le raisonneur. Elle servira de base pour l'apprentissage de nouveaux patrons d'incohérence.
- La classe *Trace_alternative* correspondant à une instance de patron d'alternatives. Elle est décrite par les propriétés de description d'un patron alternative et par la propriété *date_génération.* Elle est liée à la classe *Trace_incohérence* par la relation *résolue_par* indiquant les incohérences résolues par une instance d'alternative.
- La classe *Trace_résolution* représentée par une relation N-aire entre les classes *Trace_changement*, *Trace_incohérence* et *Trace_alternative*. Elle modélise la trace de résolution d'un changement donné causant une incohérence donnée par une alternative donnée. Elle est décrite par :
  - La propriété *date_résolution* ;
  - La propriété *type_résolution* correspondant à un type énuméré *{additionnelle, substitutive}* où la valeur *additionnelle* indique que l'instance de résolution est appliquée conjointement au changement requis et la valeur *substitutive* indique que l'instance de résolution remplace le changement requis. Cette propriété sera détaillée dans le chapitre 4 ;
  - La propriété *trace_résolution_précédente* indique la résolution précédente générée pour un changement est une incohérence.
- La classe *Trace_résolution_chgt* modélise la trace d'une résolution globale d'un changement combinant des instances d'alternatives pour résoudre toutes les incohérences causées par un changement. Elle est décrite par :
  - La propriété *date_résolution_chgt* ;
  - La propriété *cohérence* correspondant à un type booléen. La valeur *true* indique que l'instance de résolution globale de changement est cohérente, la valeur *false* indique qu'elle est incohérence ;
- La propriété *trace_résolution_chgt_précédente* indique la résolution globale précédente générée et permet de tracer – pour une transaction de changement – la succession des résolutions globales
- La classe *Trace_évaluation* modélise l'évaluation d'une instance de *Trace_résolution_chgt* cohérente. Cette restriction de valeur sur la propriété *évaluée_par* est représentée par la contrainte {*cohérence = true}*. La classe *Trace_évaluation* est décrite par :

- La propriété *date_évaluation*,
- La propriété *impact_global* correspondant à un type énuméré *{+, -, null}* indique respectivement que l'instance de résolution de changement évaluée améliore, altère ou préserve la qualité de l'ontologie,
- Les propriétés *évaluation_complexité*, … correspondant aux valeurs d'évaluation des différents critères.

- La classe *Modèle_qualité* modélise le modèle de qualité employé dans l'évaluation (les différents critères qui le définissent, leurs pondérations, etc.) à travers l'agrégation de classe *Critère_qualité* spécialisée par les classes correspondantes aux différents types de critères. Cette partie sera détaillée dans le chapitre 6.
- La classe *Trace_résolution_chgt_sélectionnée* spécialise la classe *Trace_résolution_chgt*. La spécialisation porte sur une autre restriction de valeur sur la propriété *évaluée_par*, représentée par la contrainte {*impact_global = + ou null}*. Elle correspond à une instance de résolution de changement cohérente ayant le meilleur impact sur la qualité de l'ontologie.
- La classe *Trace_évolution* représente l'historique de toutes les résolutions de changement opérées à la gestion d'une transaction de changement. Elle est décrite par :
  - La propriété *date_demande* trace la date de déclenchement de la gestion d'une demande de changement,
  - La propriété *trace_évolution_précédente* indique l'évolution précédente gérée et permet de tracer la succession des évolutions de l'ontologie,
  - La propriété *résultat_gestion_évolution* correspondant à un type énuméré défini par les valeurs :
    - *non_traitée* : indique que l'instance d'évolution correspond à un changement non traité, n'ayant pas une correspondance de spécification avec aucun des patrons de changements,
    - *non_gérée* : indique que l'instance d'évolution comprend une ou plusieurs incohérence(s) inconnues et n'a donc pas été gérée,
    - *non_résolue* : indique que les incohérences liées à l'instance d'évolution sont non résolues (les alternatives qui sont générées à partir des patrons sont inapplicables ou les résolutions de changement proposées sont toutes incohérentes.),
    - *résolutions_non_retenues* : indique que toutes les résolutions de changement liées à l'instance d'évolution sont non retenues faute d'un impact négatif sur la qualité de l'ontologie,
    - *validée :* indique qu'une résolution de changement a été sélectionnée et donc l'évolution est validée et peut être appliquée.

La classe *Trace_évolution* représente une classe transversale entre la dimension *trace d'exécution du processus pour la gestion d'une transaction de changement* et la dimension *trace d'évolution d'une version de l'ontologie*.

V.2.2- Trace d'évolution d'une version de l'ontologie

La dimension trace d'évolution d'une version de l'ontologie, est modélisée par (figure 6):

- La classe *Version_entité* modélise une version d'une entité de l'ontologie de domaine. Toute instance de *Trace_évolution* dont la gestion a été validée, est appliquée aux entités de l'ontologie modifiées par cette instance d'évolution. De nouvelles versions de ces entités sont alors créées. La nouvelle version d'une entité devient sa version courante. Cette restriction de valeur sur la

propriété *appliquée_à* est représentée par la contrainte {*résultat_gestion_évolution = validée}*. La classe *Version_entité* est décrite par :

- La propriété *date_version_entité* trace la date de création d'une instance de version d'entité,
- La propriété *id_version_entité* indique le nom de l'entité de l'ontologie dans sa version correspondante à une instance de version d'entité. La valeur de cette propriété peut être différente d'une version à une autre d'une entité de l'ontologie, si cette entité a été renommée[21],
- La propriété *version_entité_précédente* indique la version précédente à une instance de version d'entité et permet de tracer la succession des versions d'une entité.

- La classe *Ligne_de_vie_entité* décrit l'évolution chronologique d'une entité de l'ontologie de domaine de sa création à sa suppression. A la création d'une nouvelle entité dans l'ontologie de domaine, une nouvelle instance de la classe *Ligne_de_vie_entité* est créée dans le journal d'évolution. Chaque modification de cette entité sera ensuite, sauvegardée comme une nouvelle version et donc une nouvelle instance de la classe *Version_entité*. La classe *Ligne_de_vie_entité* est décrite par :
  - La propriété *date_création_entité* trace la date de création de l'entité dans une version de l'ontologie de domaine,
  - La propriété *id_entité* indique le nom de l'entité de l'ontologie à sa création.
- La classe *Version_ontologie* modélise une version de l'ontologie de domaine. Tout comme pour les entités, à l'échelle de l'ontologie, le journal d'évolution sauvegarde les différentes versions d'une ontologie de domaine. Toute demande d'application d'un changement est considérée comme une évolution de la version courante de l'ontologie. Le passage à une nouvelle version de l'ontologie ne se fait que suite à une demande explicite par l'utilisateur. Dans ce cas, une nouvelle instance de *Version_ontologie* est alors créée, elle devient la version courante de l'ontologie de domaine. Les instances de la classe *Ligne_de_vie_entité* correspondantes à toutes les entités présentes dans la version courante de l'ontologie sont réinitialisées, à chacune une première instance de la classe *Version_entité* est créée pour représenter la dernière version de l'entité correspondante. La classe *Version_ontologie* est décrite par :
  - La propriété *date_version_ontologie* trace la date de création d'une version de l'ontologie de domaine,
  - La propriété *id_version_ontologie* indique le nom de l'ontologie dans sa version correspondante. La valeur de cette propriété peut être différente d'une version à une autre de l'ontologie, si l'ontologie a été renommée,
  - La propriété *version_ontologie_précédente* indique la version précédente à une version de l'ontologie et permet de tracer la succession des versions de l'ontologie.

La classe *Version_ontologie* représente une classe transversale entre la dimension *trace d'évolution d'une version de l'ontologie* et la dimension *trace de versionning de l'ontologie*.

[21] Notons que le renommage d'une entité implique juste l'enregistrment d'une nouvelle version de cette entité sans déclencher le processus de gestion de changement. Cette opération est autorisée par tout éditeur d'ontologie mais ne correspond pas à un réel changement dans le sens sémantique du terme.

V.2.3- Trace de versionning de l'ontologie

La dimension trace de versionning de l'ontologie, est modélisée par (figure 6):

- La classe *Ligne_de_vie_ontologie* décrit le cycle de vie de l'ontologie de domaine de sa création à son abandon. La classe *Ligne_de_vie_ontologie* est décrite par :
  - La propriété *date_création_ontologie* indique la date de création de l'ontologie de domaine (ou le début du suivi de son évolution par l'approche *Onto-Evo[a]l*),
- La propriété *id_ontologie* indique le nom de l'ontologie de domaine à sa création (ou début du suivi de son évolution par l'approche *Onto-Evo[a]l*).

Ainsi, le journal d'évolution assure le suivi de l'exécution du processus pour la gestion d'une transaction de changement, le suivi de l'évolution d'une version de l'ontologie et le suivi de la succession des versions d'une même ontologie. La modélisation du journal d'évolution selon une représentation formelle et explicite des changements et des métadonnées décrivant les pratiques de gestion de changement, fournit une traçabilité détaillée de l'évolution d'ontologie et facilite le contrôle du processus d'évolution notamment pour des opérations d'annulation (ou de reprise) de changement.

VI- DISCUSSION

Le processus d'évolution *Onto-Evo[a]l* permet : (i) de spécifier formellement une demande de changement, (ii) de diriger et contrôler la gestion des changements, (iii) de maintenir la cohérence de l'ontologie et (iv) de guider les choix de résolution et optimiser l'application des changements en tenant compte de l'impact sur la qualité de l'ontologie.

L'apport de l'approche réside (1) dans la modélisation par patrons guidant le processus d'évolution, (2) l'intégration de l'évaluation de la qualité pour optimiser la résolution des changements, et (3) la modélisation formelle et explicite du journal d'évolution.

Pour discuter de l'approche *Onto-Evo[a]l* présentée dans ce chapitre, nous nous positionnons dans ce qui suit, par rapport aux travaux existants en considérant en particulier les points suivants : les niveaux de cohérence pris en compte, l'analyse et la résolution des changements, l'application des changements et la traçabilité des changements.

VI.1- Positionnement par rapport aux niveaux de cohérence pris en compte

Les conditions de cohérence d'une ontologie OWL ont été classées dans (Haase & Stojanovic, 2005), selon trois niveaux de cohérence :

- La cohérence structurelle se référant aux contraintes du langage de l'ontologie. Les auteurs proposent des résolutions d'incohérences structurelles limitées à un ensemble arbitraire de conditions structurelles de OWL Lite.
- La cohérence utilisateur ne relevant pas du langage lui-même mais se référant à des besoins spécifiques de l'utilisateur et des contraintes liées au contexte d'usage et/ou applicatif de l'ontologie. Les conditions de cohérence sont définies explicitement par l'utilisateur. Elles peuvent correspondre à des conditions de cohérence génériques permettant de préciser des critères qualitatifs de modélisation, ou à des conditions de cohérence particulières, directement liées au domaine modélisé et relevant de l'expertise du domaine.
- La cohérence logique se référant à la sémantique formelle de l'ontologie.

Onto-Evo^{a}*l* gère quatre niveaux de cohérence :

- La cohérence structurelle qui se réfère aux contraintes du langage OWL DL et à l'utilisation de ses constructeurs. Elle est assurée à travers la spécification formelle des changements à appliquer par les patrons de changements dont la modélisation s'est basée sur les fondements du modèle OWL DL ;
- La cohérence logique qui tient compte de la sémantique exprimée en DL et des interprétations au niveau instance. Elle est vérifiée et maintenue à travers tout le processus de gestion de changement de *Onto-Evo*^{a}*l* et se matérialise essentiellement par les patrons d'incohérences et les patrons d'alternatives de résolution ainsi que les relations conceptuelles entre eux ;
- La cohérence conceptuelle se référant à la conceptualisation de l'ontologie c'est-à-dire, à la manière dont elle est modélisée mais aussi modifiée/évoluée. Elle correspond à des contraintes de bonne conceptualisation. Ce niveau de cohérence est assuré d'une part, par les patrons de changements et les patrons d'alternatives correspondant à des changements dérivés (chapitre 4); et d'autre part, par un ensemble de critères définis dans le modèle de qualité ;
- La cohérence de modélisation de domaine confrontant la conceptualisation de l'ontologie au domaine qu'elle modélise pour voir si elle reflète bien les connaissances du domaine. Cette dimension est prise en compte dans le modèle de qualité à travers un ensemble de critères évaluant l'usage de l'ontologie par rapport aux sources de domaine. Ces critères seront détaillés dans le chapitre 6.

VI.2- Positionnement par rapport à l'analyse et la résolution des changements

L'analyse des effets de changements inclut non seulement la vérification de la cohérence mais aussi la résolution des incohérences détectées.

Dans (Stojanovic, 2004), un processus global d'évolution d'ontologie KAON a était proposé spécifiant la sémantique des changements KAON et assurant la cohérence à travers des stratégies de résolution basées sur les contraintes du langage KAON. Les stratégies sont présentées à l'utilisateur pour qu'il décide de leur application en se basant sur son expertise.

Tout comme plusieurs travaux en évolution d'ontologie, nous reprenons les fondements du processus global proposé pour l'évolution des ontologies KAON pour les adapter à la gestion des changements d'ontologie OWL DL dans un contexte local (non distribué). Mais la gestion des changements étant dépendante du modèle de représentation de l'ontologie, nous nous positionnons surtout par rapport aux travaux traitant l'évolution des ontologies OWL.

Les principes de résolution des incohérences KAON ont aussi été adaptés dans (Haase & Stojanovic 2005) aux ontologies OWL. Les auteurs ont introduit des stratégies de résolution basées sur les contraintes de OWL Lite. Chaque stratégie correspond à une fonction globale assignant à chaque condition de cohérence OWL Lite, une fonction de résolution permettant de générer les changements additionnels à appliquer pour la satisfaire. La résolution des incohérences logiques se base sur la détermination et la suppression des axiomes causant les incohérences selon deux approches : l'une générant le nombre minimal de changements à appliquer pour obtenir la sous-ontologie cohérente maximale et l'autre permettant de localiser les incohérences c'est-à-dire, de retrouver la sous-ontologie incohérente minimale. Dans les deux cas, les axiomes à supprimer sont ensuite présentés à l'utilisateur pour qu'il puisse décider et contrôler l'évolution de l'ontologie. Pour sélectionner les axiomes à supprimer, les auteurs proposent l'idée de définir des fonctions de sélection en s'inspirant des travaux sur l'incertitude et la réification et sur les recherches en ontologies contextuelles pour dégager des métadonnées décrivant les axiomes (confidence, degré de certitude, caractéristiques contextuelles, etc.). Mais, l'intervention de l'utilisateur reste indispensable pour décider des axiomes à supprimer.

Dans notre approche, les propositions de résolution sont faites par le système en se basant sur les patrons *CMP* et non sur des stratégies prédéfinies par l'utilisateur. La prise en compte des contraintes de l'utilisateur est intégrée dans l'évaluation de la qualité (les critères et les métriques définis dans le modèle et la pondération des critères). Ainsi, nous déterminons pour chaque changement (patron de changements), les incohérences logiques susceptibles d'être causées (patrons d'incohérences) et nous proposons pour chacune les alternatives de résolution permettant de la résoudre (patrons d'alternatives).

La localisation des incohérences se base sur la détermination de la sous-ontologie incohérente minimale. Elle sera détaillée dans le chapitre 5. Par ailleurs, pour résoudre les incohérences, nous tendons à minimiser les solutions de suppression d'axiome en proposant plutôt des patrons d'alternatives modélisant des fusions, des divisions, des généralisations ou des spécialisations de classes et ce, afin de préserver les connaissances existantes. Ensuite, nous orientons le choix des résolutions à appliquer en faisant intervenir l'évaluation de la qualité.

En effet, l'application d'un changement peut altérer non seulement la cohérence mais aussi la qualité de l'ontologie. Selon cette perspective, nous proposons une approche complémentaire aux travaux en évolution d'ontologie tenant compte de l'impact sur la qualité.

Ainsi, plutôt que de solliciter directement l'expert dans la résolution des incohérences, les résolutions générées sont évaluées par rapport à leur impact sur la qualité de l'ontologie dans le but de guider et d'optimiser la validation des résolutions. La résolution qui préserve la qualité de l'ontologie, peut être automatiquement choisie et les changements directement validés. L'expert ne sera sollicité que si toutes les résolutions ont un impact négatif sur la qualité. Il sera guidé dans sa décision, par l'ensemble des résultats d'analyse, de résolution et d'évaluation.

VI.3- Positionnement par rapport à l'application des changements

Dans (Stojanovic, 2004), l'auteur souligne l'importance de la séparation de l'analyse des changements de leur application finale. L'application réelle des changements n'est exécutée que si les résultats de l'analyse sont approuvés par l'ingénieur d'ontologie. Les changements nécessaires et dérivés sont considérés comme une transaction atomique exécutée en une seule unité dans le but de satisfaire un besoin de changement. L'accomplissement de la transaction implique l'exécution finale des changements, autrement la transaction est interrompue.

Sur le même principe, nous appliquons le changement requis et ses changements dérivés en une même transaction au cours d'un cycle du processus. Par contre, la dépendance à l'utilisateur est réduite grâce au guidage du processus par la modélisation des patrons *CMP* et à l'orientation du choix des résolutions par l'évaluation de la qualité. Ceci n'empêche qu'il ait toujours le contrôle sur le processus et puisse à tout moment, annuler, ou réajuster les pondérations attribuées aux critères de qualité, ou même modifier le choix des résolutions – retenues par le système – en négligeant l'impact sur la qualité.

VI.4- Positionnement par rapport à la traçabilité des changements

Deux questions sont considérées dans cette section : le format de modélisation du journal d'évolution et la traçabilité des niveaux entité, version d'une ontologie et cycle de vie d'une ontologie.

VI.4.1- Format de modélisation du journal d'évolution

Dans (Maedche et al., 2002), deux notions ont été introduites :

- L'ontologie d'évolution définit un modèle des changements applicables à une ontologie de domaine KAON. Elle sert de base pour le processus d'évolution et pour la construction du journal d'évolution ;

- Le journal d'évolution exprimé en XML/RDF, permet d'enregistrer la séquence chronologique des opérations appliquées à une ontologie KAON pour satisfaire une demande de changement (Stojanovic, 2004). Les informations enregistrées correspondent à des instances des concepts et des propriétés de l'ontologie d'évolution.

L'ontologie d'évolution de Stojanovic se rapproche de l'ontologie *CMP* particulièrement au niveau de la description des patrons de changements OWL DL. Mais l'ontologie *CMP* rajoute une modélisation explicite des incohérences pouvant être causées par les changements (patrons d'incohérences), des alternatives pouvant les résoudre (patrons d'alternatives), et des liens conceptuels reliant des types de changements, d'incohérences et d'alternatives offrant ainsi, un vrai guidage du processus d'évolution.

Le journal d'évolution dans l'approche *Onto-Evo*^{a}*l* est présenté sous forme d'une ontologie OWL DL, ce qui offre un format explicite, plus formel que le format XML/RDF compatible avec le modèle des ontologies de domaine gérées et celui de l'ontologie *CMP* guidant le processus (figure 5). La conceptualisation de l'ontologie journal d'évolution inclut des propriétés temporelles décrivant la chronologie des changements demandés et des changements intermédiaires appliqués, les détails de traitement de ces changements et les versions des entités de l'ontologie de domaine ainsi que de l'ontologie elle-même.

VI.4.2- Traçabilité des niveaux entité, version d'une ontologie et cycle de vie d'une ontologie

Plusieurs travaux se sont penchés sur la traçabilité d'entités supprimées. Dans (Oliver et al., 1999), les auteurs discutent des changements appliqués aux ontologies médicales et proposent de définir un état « *retiré* » aux concepts supprimés plutôt que de les supprimer physiquement et de maintenir certains liens entre concepts pour assurer par exemple, la traçabilité des concepts parents et sous-concepts retirés d'un concept. Cette approche a aussi été adoptée pour des entrepôts à base ontologique (Xuan et al., 2006). Cependant, elle alourdit le contenu de l'ontologie en gardant des entités supprimées, et complique sa lisibilité. De plus, elle ne répond pas aux besoins d'évolution d'ontologie, dans le sens d'un processus contrôlé de gestion de changement, d'une concrète réalisation du changement et d'une traçabilité des évolutions dans le temps.

Dans les travaux de (Plessers et al., 2007), les auteurs se focalise sur la trace d'évolution des entités de l'ontologie. Le log d'évolution de l'ontologie – appelé *historique de version* – sauvegarde l'historique d'évolution de chaque entité définie dans l'ontologie. Un historique d'évolution d'une entité correspond à la ligne de vie de l'entité, qui commence à sa création, décrit ses différentes modifications et continue jusqu'à la fin de son cycle de vie (sa suppression de l'ontologie). Les différents états d'une ontologie dans le temps sont capturés par des *snapshots* (photos instantanées) sur les définitions de ses entités, basés sur la logique temporelle.

Notre approche intègre l'idée de capturer l'évolution à l'échelle des entités de l'ontologie. Elle est modélisée dans l'ontologie journal d'évolution, par la classe *Ligne_de_vie_entité* correspondant dans les travaux de (Plessers et al., 2007), à une ligne de temps linéaire et discrète exprimée en logique temporelle. Les différentes versions d'une entité sont modélisées par la décomposition de la classe *Ligne_de_vie_entité* en classe *Version_entité*. Une instance de la classe *Version_entité* correspond, dans les travaux de (Plessers et al., 2007), à un « *timepoint* » défini en logique temporelle sous forme d'un quadruplet composé du nom de l'entité dans cette version, de l'ensemble des axiomes décrivant sa définition en logique de description, de la date de début et de la date de fin de cette version. Outre l'évolution des entités, le journal d'évolution de *Onto-Evo*^{a}*l* modélise aussi l'évolution d'une ontologie le long de son cycle de vie et l'évolution de ses différentes versions à travers la décomposition de la classe *Ligne_de_vie_ontologie* en classe *Version_ontologie*.

Un autre point de discussion s'impose à ce niveau, c'est celui de la taille du journal d'évolution et du problème de capacité de stockage. Ce problème est présenté dans les travaux de (Plessers et al., 2007) comme étant justement l'argument justifiant le choix de représenter dans le log, des *snapshots* sur les définitions des entités plutôt que sur l'ontologie entière. L'ontologie journal d'évolution que nous avons définie, optimise le stockage à l'échelle d'une version de l'ontologie, puisque à chaque modification, seules les lignes de vie des entités concernées sont mises à jour. Mais, l'enregistrement des détails de gestion des changements et la prise en compte de différentes versions de l'ontologie, augmentent considérablement les besoins en espace de stockage.

Rappelons que les travaux de (Plessers et al., 2007) ont été menés dans le cadre d'une approche de détection de changements déjà appliqués. Le log a pour rôle de faciliter la réponse à des requêtes de détection de changement ciblant l'évolution d'une entité. Les objectifs d'usage du log d'évolution ne sont les même dans l'approche *Onto-Evo[a]l.* La modélisation de l'ontologie journal d'évolution répond à l'un des principaux objectifs de l'approche *Onto-Evo[a]l* qui est de faire du journal d'évolution, une base de connaissances qui offre une traçabilité détaillée des entités (ajoutées, modifiées, renommées ou supprimées) et des traitements de gestion de changement de ces entités, facilitant le contrôle du processus d'évolution et permettant d'enrichir le méta-modèle guidant le déroulement du processus – les patrons *CMP* – par l'apprentissage de nouveaux patrons. De plus, elle permet de conserver les relations entre les versions successives de l'ontologie de domaine. Cette dimension « versionning de l'ontologie » est intégrée au modèle du journal pour permettre la gestion des versions en cas de besoin, mais n'est pas opérée de manière automatique (pas de passage de version sauf par demande explicite par l'utilisateur). Il est aussi possible d'envisager en cas de gestion de plusieurs versions de l'ontologie, une solution d'archivage des versions les plus anciennes de l'ontologie – qui sont devenues obsolètes – et de ne garder dans le journal que les dernières versions qui sont plus représentatives pour le contexte d'application de l'ontologie.

VII- CONCLUSION

Dans ce chapitre, nous avons présenté l'approche d'évolution *Onto-Evo[a]l* (Ontology Evolution-Evaluation) dont le processus de gestion de changement assure :

- L'analyse et l'explication des changements et de leurs impacts en explicitant tout changement demandé ainsi que l'ensemble des modifications permettant de le réaliser (phase de spécification) et ses conséquences sur la cohérence logique (phase d'analyse) ;
- L'application assistée et optimisée des changements et la justification des traitements en générant des solutions de résolutions assurant la cohérence de l'ontologie évoluée tout en préservant, si possible, la qualité.

La principale contribution de l'approche est d'intégrer une modélisation par patrons dans un processus de gestion des changements d'ontologies permettant de définir et de classer des types de changements, d'incohérences et d'alternatives et de faire ressortir des liens entre ces patrons pour guider et contrôler l'analyse et la résolution des impacts de changements. De plus, l'évaluation de l'impact sur la qualité permet de mieux conduire la résolution des changements et de préserver la qualité de l'ontologie évoluée.

La description détaillée de la modélisation des patrons de gestion de changement *CMP* (Change Management Patterns) fait l'objet du chapitre suivant.

Chapitre 4 : CMP : Patrons de gestion de changement d'ontologie

Résumé : Dans ce chapitre, nous détaillons la modélisation des patrons de gestion de changement *CMP* (Change Management Patterns) et les liens conceptuels entre eux. Ces patrons correspondent à des types de changements, d'incohérences et d'alternatives de résolution gérés par l'approche *Onto-Evo*^{a}*l* et spécifiés formellement par l'ontologie *CMP*. Ils sont employés dans le processus de gestion de changement comme des méta-connaissances offertes pour guider et contrôler l'application des changements tout en assurant la cohérence logique de l'ontologie évoluée.
Les patrons CMP sont représentés selon deux couches : une couche de présentation sous forme d'un catalogue de patrons décrits en langage naturel et illustrés par des diagrammes UML, et une couche de spécification formelle sous forme d'une ontologie OWL DL définissant la sémantique des patrons, des relations entre eux, et de leur application.

I- INTRODUCTION

L'approche d'évolution *Onto-Evo*^{a}*l* (Ontology Evolution-Evaluation) que nous avons définie (Chapitre 3), s'appuie sur une modélisation à l'aide de patrons : les patrons de gestion de changement *CMP* (Change Management Patterns). Ces patrons spécifient des classes de changements, des classes d'incohérences logiques et des classes d'alternatives de résolution. Sur la base de ces patrons et des liens conceptuels définis entre eux, nous proposons un processus automatisé permettant de conduire l'application des changements tout en maintenant la cohérence de l'ontologie évoluée.

La gestion des changements étant fortement liée au modèle dans lequel est représentée l'ontologie, la modélisation des patrons ainsi que leur implémentation sont basées sur la syntaxe et la sémantique du modèle OWL DL telles que spécifiées par le W3C[22].

Le chapitre est organisé comme suit : dans la section 2, nous présentons une étude synthétique de l'état de l'art en modélisation dirigée par patrons particulièrement pour la conception d'ontologies. Dans la section 3, nous décrivons le modèle conceptuel général des patrons *CMP* et le format adopté pour leur présentation. Les différents types de patrons *CMP* – patrons de changements, d'incohérences et d'alternatives – sont détaillés et illustrés dans la section 4. La section 5 expose notre démarche de construction de l'ontologie *CMP* et décrit l'ontologie. Avant de conclure le chapitre, une discussion est présentée en section 6.

II- MODELISATION DIRIGEE PAR LES PATRONS

II.1- Historique

Dans la littérature, les patrons sont abordés soit selon une approche de découverte de structures régulières et réutilisables dans des implémentations existantes ; soit selon une approche d'emploi de patrons prédéfinis pour la réutilisation de modèles, de connaissances ou de composants implémentés, dans le but de faciliter des processus de construction de modèles conceptuels, de base de connaissances, ou de

[22] http://www.w3.org/TR/owl-semantics/

logiciels. C'est sur la deuxième approche que nous nous focalisons pour présenter un état de l'art sur l'utilisation de patrons pour la conception d'ontologie et par là, introduire les patrons de gestion de changement d'ontologie CMP que nous avons définis.

Il existe différents types de patrons selon les différents domaines dans lesquels ils ont été utilisés :

- En ingénierie logicielle, les patrons logiciels sont essentiellement utilisés en développement orienté objet. Ils facilitent le contrôle de la complexité et de la qualité des logiciels développés ainsi que la réutilisation. Ils renferment entre autres des idiomes de langages de programmation (Buschmann et al., 1996), des patrons d'analyse (Fernandez & Yuan, 2000) (Kolp et al., 2003), des patrons de conception (Gamma et al., 1995) (Buschmann et al., 1996) (Hunt, 2003) et des patrons d'architecture (Buschmann et al., 1996) (Shaw, 1996) (Fowler, 2003) ;
- En modélisation de données, les patrons de modèles de données proches des patrons logiciels et pouvant en faire partie (Hay, 1996) ;
- En linguistique, les patrons linguistiques représentent la structure d'un langage. Ils sont très utilisés par les outils TALN notamment pour l'extraction de connaissances à partir de textes (Maedche & Volz, 2001) (Aussenac-Gilles & Jacques, 2006) (Jacques & Aussenac-Gilles, 2006) et l'identification de relations sémantiques (Chagnoux et al., 2008) ;
- En modélisation de connaissances, les patrons sémantiques ont pour objectif d'abstraire des représentations de structure ou de langage tout en modélisant leur sémantique. Ils constituent des métalangages et sont notamment utilisés en ingénierie ontologique (Staab et al., 2001) ;
- En gestion de connaissances (Knowledge Management), les patrons de connaissances correspondent à des patrons de construction de bases de connaissances (Clark et al., 2000), des patrons de réutilisation de méthodes de résolution de problèmes (Puppe, 2000), ou encore – dans le domaine business – à des patrons d'organisation de connaissances d'entreprises (Persson & Stirna, 2002) (Stirna & Persson, 2002).

II.2- Application aux ontologies : les patrons de conception d'ontologies ODP

La modélisation par patrons a été adoptée ces dernières années, dans la conception d'ontologies pour le Web afin de proposer des guides de bonnes pratiques et de fournir des catalogues de composants ontologiques réutilisables[23]. La notion de patron de conception d'ontologies a été introduite par (Clark et al., 2000) (Gangemi et al., 2004), (Rector & Roger, 2004), (Svatek, 2004). Depuis, différents travaux de recherche (Pinto et al., 2004) (Guizzardi, 2005), (Gangemi, 2005), (Gangemi & Presutti, 2007) (Presutti & Gangemi, 2008) et d'efforts de développement ont été menés dans ce but notamment avec le groupe *W3C OEP Task Force*[24].

Les patrons de conception d'ontologie *ODP* (Ontology Design Patterns) sont conçus comme des directives (guidelines) partagées et réutilisables dans un contexte de conception collaborative d'ontologies en réseau (Presutti et al., 2008). Ils ont été proposés dans l'objectif d'offrir à la communauté du web sémantique, un espace d'apprentissage, de partage et de discussion sous forme d'un portail sémantique[25] muni d'un référentiel de patrons de conception d'ontologie (Daga et al., 2008).

Le portail ODP correspond à un wiki développé sous forme d'une extension de MediaWiki[26] pour répondre aux besoins des patrons de conception d'ontologie, notamment en termes de validation des patrons déposés sur le portail, à travers un workflow qui inclut une étape d'évaluation par les membres du

[23] Exemple : http://odps.sourceforge.net/
[24] Semantic Web Best Practices and Deployment Working Group. Ontology Engineering and Patterns Task Force (OEP). http://www.w3.org/2001/sw/BestPractices/OEP/
[25] http://ontologydesignpatterns.org/wiki/Main_Page
[26] http://www.mediawiki.org

conseil éditorial du portail – dont nous faisons partie – suivie d'une étape de certification des patrons retenus par les membres du comité qualité.

Les patrons d'ontologie ODP apportent des solutions réutilisables à des problèmes récurrents de modélisation d'ontologie. Ils sont classés selon différents types[27] (Presutti et al., 2008) :

- Les patrons de contenu sont des patrons conceptuels réutilisables visant à construire des ontologies valides par rapport à un domaine et facilement exploitables du moins, en tant que vocabulaire de référence. Ils résolvent des problèmes de conception liés aux classes et propriétés du domaine modélisé (Gangemi, 2005) (Presutti & Gangemi, 2008). Dans (Blomqvist, 2009), un Framework de construction et de raffinement semi-automatiques d'ontologies guidés par les patrons est présenté. Le raffinement s'applique à des ontologies dites « légères » en termes d'expressivité et de complexité logique. Les patrons sont utilisés pour définir automatiquement le contexte auquel se réfèrent des éléments ontologiques, obtenus par apprentissage, afin de guider leur intégration dans l'ontologie de domaine. Les éléments ontologiques appris sont mis en correspondance avec les concepts et les relations les plus génériques des patrons. La sélection des patrons se base essentiellement sur des matchings terminologiques entre les labels des concepts et des relations définis par un patron, et les termes désignant les éléments ontologiques appris à partir de sources.
- Les patrons de structure :
  - Les patrons logiques sont indépendants du domaine modélisé par l'ontologie et donc, de toute conceptualisation particulière. Ils correspondent à des compositions de constructions logiques permettant de résoudre des problèmes d'expressivité (Presutti et al., 2008),
  - Les patrons d'architecture concernent la forme globale de l'ontologie et permettent de guider les choix de conception en fonction de besoins spécifiques (Gangemi et al. 2007),
- Les patrons de correspondance :
  - Les patrons de réingénierie correspondent à des règles de transformation de modèles ontologiques ou non (thésaurus, un modèle UML, etc.) en ontologies. Des travaux spécifiques ont été présentés dans (Garcia-Silvia et al., 2008),
  - Les patrons d'alignement permettent de créer des associations sémantiques entre deux ontologies modélisant des domaines similaires. Ils modélisent des types fréquents d'alignement. Des travaux spécifiques aux patrons de correspondance (dans le sens alignement) ont été présentés dans (Scharffe, 2009),
- Les patrons de raisonnement correspondent à des applications de patrons logiques ciblant des résultats de raisonnement particuliers. Ils peuvent correspondre à des classifications, des subsomptions, etc. ;
- Les patrons de présentation se réfèrent à l'utilisabilité et la facilité de compréhension de l'ontologie :
  - Les patrons de nommage se réfèrent à l'homogénéité dans les procédures de nommage (notamment la déclaration des espaces de nommage) pour faciliter la lisibilité de l'ontologie ;
  - Les patrons d'annotation offrent des propriétés et des schémas de propriétés d'annotation facilitant la compréhension des entités de l'ontologie.
- Les patrons lexico-syntaxiques correspondent à des structures et des schémas linguistiques facilitant l'extraction et /ou la traçabilité des connaissances par rapport aux sources textuelles.

L'implémentation des patrons ODP est exprimée en OWL.

[27] http://ontologydesignpatterns.org/wiki/OPTypes

II.3- Différents domaines d'application

Les patrons de conception d'ontologie sont appliqués dans différents domaines :

- Construction d'ontologie d'entreprise (Blomqvist, 2005) ;
- Contenu objets et multimédia (Arndt et al., 2007) ;
- Peuplement et enrichissement d'ontologie multimédia dans la méthodologie BOEMIE[28] (Castano et al., 2007) présentée dans le chapitre 2 (section III.2) ;
- Composants logiciels (Oberle, 2006) ;
- Modélisation et interaction business (Guizzardi & Wagner, 2004) (Gangemi & Presutti, 2007) ;
- Conception de processus business (Damjanovic, 2009) ;
- Pertinence de conception en OWL (Gomez-Romero et al., 2007) ;
- Conception d'ontologies biologiques (Aranguren et al., 2007) ;
- Conception d'ontologies juridiques (Gangemi, 2007).

II.4- Les initiatives les plus récentes

Avant de clore cette présentation succincte de l'état de l'art en modélisation par les patrons, nous nous proposons de citer deux des dernières initiatives de définition et de dissémination de bonnes pratiques et de modélisation de patrons pour la construction d'ontologies, une à l'échelle nationale et une à l'échelle internationale :

- L'atelier de construction d'ontologies : Vers un guide de bonnes pratiques[29], organisé dans le cadre de la plateforme AFIA 2009 conjointement avec la conférence IC 2009. Cet atelier a permis de partager des expériences tirées de la construction de diverses ontologies avec l'objectif d'éditer et de diffuser un guide de bonnes pratiques. Les principaux points abordés concernent les méthodologies de construction et le développement pragmatique d'ontologies, les approches de construction à adopter en fonction du type des sources, les acteurs et les outils intervenant dans les différentes étapes du cycle de développement, l'objectif applicatif d'une ontologie et son utilisabilité, la réutilisation d'ontologies existantes, etc.
- Le Workshop de patrons d'ontologies WOP2009[30] (Workshop on Ontology Patterns), organisé dans le cadre de la conférence internationale ISWC 2009 par l'équipe des patrons de conception d'ontologie ODP[31]. L'objectif du workshop est de proposer et discuter de bonnes pratiques, de patrons de conception d'ontologies et de systèmes et ontologies basés sur des patrons. Outre la publication d'articles scientifiques, le workshop inclut une session de soumission de patrons d'ontologie qui seront évalués et validés sur le portail ODP.

III- MODELE CONCEPTUEL ET MODELE DE PRESENTATION DES CMP

Dans cette section, nous présentons le modèle conceptuel qui décrit les patrons de gestion de changement CMP et les liens conceptuels entre eux et nous décrivons le formalisme adopté pour leur présentation.

[28] http://www.boemie.org/
[29] http://www-limbio.smbh.univ-paris13.fr/GBPonto/?page_id=2
[30] http://ontologydesignpatterns.org/wiki/WOP2009:Main
[31] http://ontologydesignpatterns.org/wiki/Main_Page

III.1- Modèle conceptuel des CMP

Tout comme l'idée de modéliser des patrons de conception pour la construction d'ontologies OWL (Gangemi et al., 2007), les patrons *CMP* sont proposés comme une solution permettant de faire ressortir des invariances observées répétitivement lors d'un processus d'évolution d'ontologie. De la modélisation des CMP en ressort trois catégories de patrons : des *patrons de changements*, des *patrons d'incohérences* et des *patrons d'alternatives* de résolution. L'intérêt de cette modélisation est d'offrir différents niveaux d'abstraction pour ces trois dimensions modélisées, d'établir des liens entre ces trois catégories de patrons déterminant les types d'incohérences qui peuvent être potentiellement causées par un type de changement et les types d'alternatives possibles pour ces incohérences et le changement qui les a causées, et par là, d'assurer un processus automatisé de gestion de changement. Le modèle conceptuel des patrons CMP est décrit par la figure ci-après (figure 1).

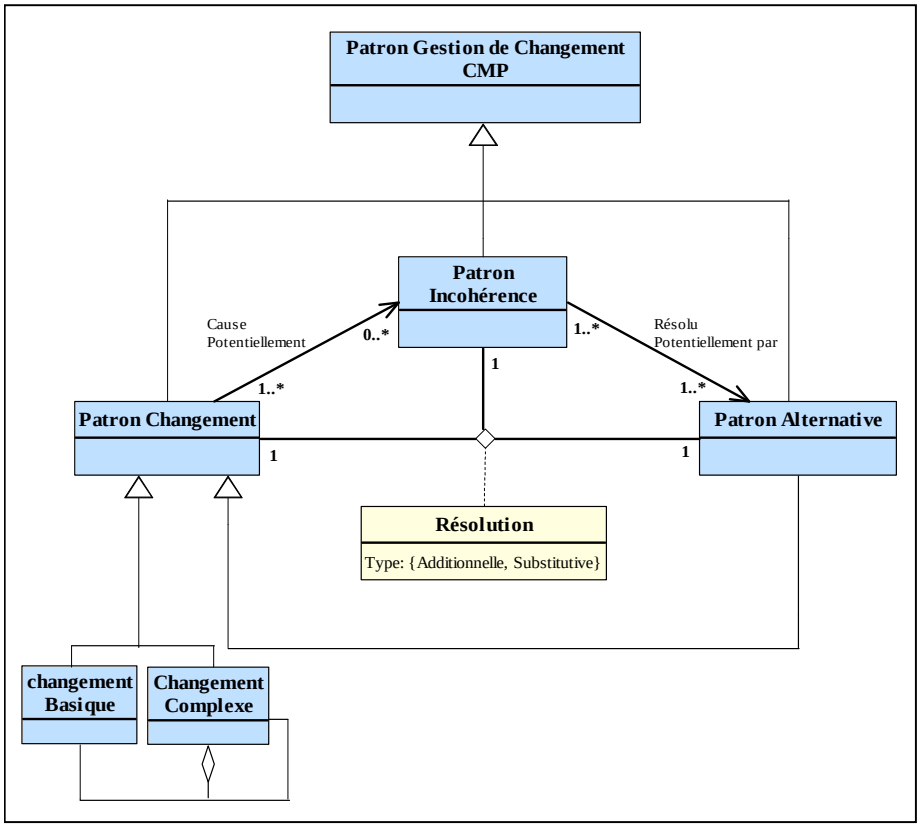


Figure 1. Modèle conceptuel des patrons de gestion de changement CMP.

Ainsi, un *Patron CMP* est un *Patron Changement*, ou un *Patron Incohérence* ou encore un *Patron Alternative*. La classe *Résolution* modélise une alternative de résolution pour une incohérence et un changement donnés. Elle peut être soit additionnelle (*Patron Alternative* s'applique conjointement au *Patron Changement*) soit substitutive (*Patron Alternative* remplace *Patron Changement*).

La relation *Cause Potentiellement* permet de prévoir les types d'incohérences attendus par l'application d'un changement – avant le lancement du raisonneur – afin d'orienter la détection des incohérences à la phase d'analyse de changement du processus *Onto-Evo*^{a}*l*.

La relation *Résolu Potentiellement par* permet, pour une incohérence détectée non attendue, de proposer des alternatives indépendantes du type du changement.

Un *Patron Changement* peut correspondre à un Patron *Changement Basique* ou à un Patron *Changement Complexe* tel que spécifié dans la classification des changements OWL (Klein, 2004) (voir chapitre 2, section II.3.3). Un patron *Changement Complexe* est composé d'un ensemble de patrons *Changement Basique* et/ou *Changement Complexe*.

III.2- Formalisme de présentation des CMP

Une modélisation à l'aide de patrons a pour objectif premier de fournir des directives partagées et réutilisables. Il est donc nécessaire de penser à un format de présentation simple, lisible, compréhensible et documenté facilitant la diffusion et la réutilisation des patrons définis. Par ailleurs, afin de réaliser entièrement leur rôle de guidage du processus de gestion de changement, les patrons CMP doivent être représentés dans un formalisme suffisamment formel pour faciliter une interprétation explicite de leur sémantique qui soit aussi, compréhensible par la machine si nous souhaitons automatiser le processus d'évolution. De plus, l'implémentation du système qui utilisera et appliquera ces patrons, se trouve facilitée par l'homogénéisation des modèles de connaissances qu'il manipule et donc en assurant une compatibilité avec le modèle des ontologies de domaine dont le système gère l'évolution : le modèle OWL DL. Pour toutes ces raisons, nous avons choisi de représenter les patrons CMP selon deux couches :

- Une couche de présentation permettant de les présenter sous forme d'une librairie partagée de patrons de gestion de changement *CMP*, de les documenter, de les diffuser et même de les illustrer. Cette couche est présentée dans cette section ;
- Une couche de spécification formelle qui permet de les employer pour le guidage du processus de gestion de changement : l'ontologie CMP exprimée en OWL DL. Cette ontologie est présentée dans la section 5.

Le formalisme choisi pour la présentation des CMP étend le formalisme de représentation des patrons de conception d'ontologie ODP (Ontology Design Patterns) (Gangemi et al., 2007), lui-même adopté du format standard défini pour la représentation des patrons de conception en génie logiciel (Gamma et al., 1995).

La présentation des patrons CMP est organisée selon trois niveaux d'abstraction :

- Un niveau générique décrivant les propriétés des CMP communes aux trois types de patrons : changement, incohérence et alternative ;
- Un niveau spécifique décrivant les propriétés spécifiques aux trois types de patrons ;
- Et un niveau encore plus spécifique détaillant les propriétés des deux types de patrons de changements : changement basique et changement complexe.

Le modèle de présentation des CMP est un modèle (Template) à base de cadre (Frame), constitué des propriétés (slots) suivantes[32] :

- Propriétés générales : *Informations générales* (General Information) sur le patron et *Cas d'utilisation* du patron (Use Case) ;
- Propriétés spécifiques au niveau d'abstraction CMP incluant des propriétés générales de *Description du patron* (Pattern Description) – certaines indépendantes du type du patron et d'autres étendues aux différentes catégories de patrons – des propriétés de *Représentation graphique* (Graphical Representation) des patrons présentées dans un format proche du modèle

[32] La désignation en anglais des propriétés permet de retrouver – s'il y en a – les correspondances avec les propriétés de description des patrons ODP.

UML, et des propriétés décrivant l'*Implémentation* du patron en code OWL DL (Implementation);

- Propriétés de liens entre patrons décrivant les *Relations* (Relationships) d'un patron avec d'autres patrons incluant notamment la description de ses *Relations avec d'autres CMP* (Relations to Other CMP) ;
- Propriété commentaire permettant d'annoter librement le patron.

Certaines de ces propriétés incluent aussi des propriétés d'illustration par des exemples. Les différents modèles de présentation sont décrits par les tables suivantes (Table. 1) (Table. 2) (Table. 3) (Table. 4) (Table. 5).

MODELE GENERIQUE CMP	
Propriétés	**Valeur**
Propriétés générales	
Information générales	
Nom*[33]	Un nom significatif se référant au patron.
Identifiant*	Un identifiant textuel unique correspondant à un acronyme du patron (composé du type du patron + l'abréviation de son nom + un nombre séquentiel[34]).
URI	L'identifiant du patron sur le web[35].
Type CMP *	Le type du patron CMP: patron changement, patron incohérence ou patron alternative.
Alias	Des noms alternatifs possibles pour le patron.
Cas d'utilisation	
Problème	Description en langage naturel du problème abordé par le patron.
Questions de Compétence	Les questions auxquelles répond le patron. Elles expliquent le contexte dans lequel le patron peut être utilisé.
Exemples	Description en langage naturel de certains exemples du problème abordé par le patron.
Niveau d'abstraction CMP	
Description du patron	
Intention	Description en langage naturel des objectifs du patron.
Conséquences	Description en langage naturel des avantages et éventuels compromis attendus de l'utilisation du patron.
Scénarios	Exemples d'instanciation du patron exprimés en langage naturel.
Représentation graphique	
Diagramme	Représentation graphique du patron (format UML par défaut).
Exemples de diagramme	Représentation graphique de certains exemples d'instanciation du patron.
Implémentation	
Composant OWL DL réutilisable	Indique l'URI du fichier contenant le composant OWL DL implémentant le patron. Il correspond à la description ontologique du patron extraite de l'ontologie CMP.
Exemples (fichiers OWL)	Indique des liens vers des fichiers OWL implémentant certains exemples d'instanciation du patron.

[33] * indique une propriété obligatoire.
[34] Peut servir en cas de plusieurs versions du patron
[35] Propriété prévue pour une mise en disponibilité future sur le portail ODP.

Relations	
Avoir composants	Références à des patrons du même type composant le patron.
Spécialisation de	Références à des patrons du même type spécialisés par le patron. Il peut s'agir aussi d'un patron de changements que le patron d'alternative décrit spécialise.
Relations avec ODPs	Références aux patrons ODP[36] auxquels est lié le patron.
Relations avec d'autres CMP	
Patrons changement	Références aux patrons de changement liés au patron.
Patrons incohérence	Références aux patrons d'incohérence liés au patron.
Patrons alternative	Références aux patrons d'alternative liés au patron.
Commentaires	
Commentaires	Remarques et commentaires libres clarifiant l'utilisation du patron. Pour les patrons de changements, cette propriété peut servir d'annotation textuelle à la définition du changement requis pour donner les raisons de ce changement (pour le domaine ou la structure de l'ontologie, pour corriger l'ontologie, etc.).

Table 1. Modèle générique de présentation des CMP.

Dans le modèle spécifique aux patrons de changements, la propriété *Description du patron* du modèle générique, est enrichie par la sous-propriété *Arguments* (Arguments) regroupant l'ensemble des paramètres nécessaires à l'application d'un changement (son contenu varie selon le type du changement et le type des entités concernées) et la sous-propriété *Contraintes* (Constraints).

MODELE SPECIFIQUE AUX PATRONS DE CHANGEMENTS	
Niveau d'abstraction changement	
Description du patron	
Type patron de changement *	Type du patron de changement : changement basique ou changement complexe.
Type de l'objet de changement *	Type de la primitive conceptuelle sur laquelle porte le changement : classe, propriété, ou instance.
Type des entités concernées *	Type des entités concernées par le changement (classe, collection de classes, collection d'instances, restriction de valeur, etc.).
Arguments*[37]	
Objet*	Référence de l'objet sur lequel porte le changement (son ID dans l'ontologie de domaine).
Exemple	Instanciation de l'objet de changement en se basant sur un exemple.
Entités référencées*	Références des entités couvertes par le changement (leurs IDs dans l'ontologie de domaine).
Exemples	Instanciation des entités référencées par le changement en se basant sur un exemple.
Entités intermédiaires	Références (IDs) des entités intermédiaires impliquées dans l'application du changement (telles que les superclasses d'une classe à ajouter).

[36] Propriété prévue pour une mise en correspondance future avec des patrons ODP sur le portail ODP.
[37] Les sous-propriétés décrivant les arguments d'un changement diffèrent en fonction des spécificités de chaque patron. Elles peuvent être elles-mêmes décomposées en d'autres sous-propriétés.

Exemples	Instanciation des entités intermédiaires impliquées dans le changement en se basant sur un exemple.
Valeurs du changement	Les valeurs du changement – s'il y en a – (telles que les valeurs d'une restriction de cardinalité).
Exemples	Exemple des valeurs d'un changement en se basant sur un exemple.
Contraintes	
Contraintes	Tout changement peut causer ou non une incohérence logique. La spécification d'un changement ne peut imposer des restrictions sur le changement lui-même ni évaluer par anticipation ses effets. Cela dépend du contexte d'application du changement (contenu de l'ontologie cible, emplacement du changement dans la structure de l'ontologie, etc.). Ce problème ne peut être réellement géré qu'à l'exécution de l'analyse du changement. Mais y faire face dès le stade de spécification et préparer le terrain à l'analyse, nous proposons de modéliser dans les propriétés d'un patron de changement, les contraintes à satisfaire pour que le changement puisse être appliqué sans altérer la cohérence logique de l'ontologie. Ce sont des pré-conditions qui peuvent être vérifiées avant l'application du changement pour préparer la prévision des incohérences pouvant être causées par le changement. Elles sont exprimées en OWL DL.
Exemples	Instanciation des contraintes en se basant sur un exemple.

Table 2. Modèle spécifique de présentation des patrons de changements.

Dans le modèle spécifique aux patrons de changements complexes, la propriété *Description du patron* étend celle des patrons de changements par la sous-propriété *Séquence* (Sequence). Le modèle de patrons de changements basiques n'a pas de propriétés spécifiques.

MODELE SPECIFIQUE AUX PATRONS DE CHANGEMENTS COMPLEXES	
Niveau d'abstraction changement complexe	
Description du patron	
Séquence*	Liste ordonnée de patrons de changements basiques et /ou complexes composant le patron. Les valeurs des arguments de chaque patron composant sont aussi données[38].
Exemples	Instanciation de la séquence d'un patron de changement complexe donné en exemple.

Table 3. Modèle spécifique de présentation des patrons de changements complexes.

Dans le modèle spécifique aux patrons d'incohérences, la propriété *Description du patron* du modèle générique, est enrichie par les sous-propriétés *Entités impliquées* (Implicated entities), *Entités concernées* (Involved Entities) et *Axiomes concernés* (Involved Axioms).

[38] Cette propriété est plus précise que la propriété *Relation avec d'autres CMP – Patrons changement* comme elle décrit une relation de composition séquentielle définissant le changement complexe lui-même.

MODELE SPECIFIQUE AUX PATRONS D'INCOHERENCES	
Niveau d'abstraction incohérence	
Description du patron	
Arguments*	
Entités impliquées*	Les références (IDs) de toutes les entités impliquées directement ou indirectement dans l'incohérence logique. Ces informations permettent d'expliquer l'incohérence et facilitent sa localisation.
Exemples	Instanciation des entités impliquées dans l'incohérence en se basant sur un exemple.
Entités concernées *	Les références (IDs) des entités concernées par l'incohérence. Ces informations facilitent la détermination des axiomes causant l'incohérence instance du patron (réellement détectée) et préparent la proposition de résolution.
Exemples	Instanciation des entités concernées par l'incohérence en se basant sur un exemple.
Axiomes*	
Axiomes concernés	Les axiomes concernés par l'incohérence.
Exemples	Instanciation des axiomes concernés par l'incohérence en se basant sur un exemple.
Axiomes responsables	Les axiomes causant l'incohérence.
Exemples	Instanciation des axiomes causant l'incohérence en se basant sur un exemple.

Table 4. Modèle spécifique de présentation des patrons d'incohérences.

Dans le modèle spécifique aux patrons d'alternatives spécialisant le modèle spécifique aux patrons de changement, la propriété *Description du patron* étend celle des patrons de changements par les sous-propriétés *Pré-conditions* (Preconditions) et *Processus* (Process).

MODELE SPECIFIQUE AUX PATRONS D'ALTERNATIVES	
Niveau d'abstraction alternative	
Description du patron	
Pré-conditions	Les pré-conditions à satisfaire pour choisir le patron comme solution de résolution.
Exemples	Instanciation des pré-conditions en se basant sur un exemple d'alternative.
Processus	Liste ordonnée d'actions décrivant en langage naturel le processus d'application de l'alternative pour résoudre une incohérence.
Exemples	Instanciation du processus en se basant sur un exemple d'alternative.

Table 5. Modèle spécifique de présentation des patrons d'alternatives.

IV- LES PATRONS CMP

Les patrons CMP sont introduits dans l'approche *Onto-Evo*^{a}*l* comme un moyen de faciliter l'application d'un changement tout en assurant la cohérence de l'ontologie évoluée. Ils modélisent des structures récurrentes de changements en se basant sur le méta-modèle OWL DL, des types d'incohérences logiques que ces changements peuvent potentiellement causer en considérant les contraintes de cohérence de OWL DL, et des types d'alternatives pouvant résoudre ces incohérences.

Dans cette section, nous détaillons les différents types de patrons CMP définis et nous décrivons comment se fait l'instanciation des trois catégories de patrons et des relations conceptuelles entre eux (figure 1) pour guider le processus de gestion de changement en illustrant par un exemple appliqué comme un fil conducteur, les différentes étapes de spécification, d'analyse et de résolution de changement.

IV.1- Les patrons de changements

L'application d'un changement à une ontologie permet d'ajouter une/de nouvelle(s) connaissance(s), de supprimer une/des connaissance(s) existante(s) (certaines connaissances peuvent en effet, être réfutées à un moment donné, notamment lorsque l'ontologie est en cours de construction), ou de modifier l'interprétation d'une ou de plusieurs connaissances de sorte que de nouvelles règles ou contraintes prennent effet. L'objectif visé par la modélisation des patrons de changements est de catégoriser les changements, définir formellement leur signification, leur portée et leurs implications potentielles.

Comme dans la plupart des langages d'ontologies, les primitives conceptuelles de base de OWL sont les classes, les propriétés et les individus. Elles sont organisées en hiérarchie. Cependant la sémantique de définition de ces primitives et des relations entre-elles à travers des axiomes et des compositions d'axiomes est propre au langage OWL et est encore plus spécifique dans sa couche OWL DL. C'est pourquoi, la modélisation de patrons de changements – même si elle est suffisamment générique dans la classification de types de changements, pour être applicable à d'autres langages d'ontologies – est formellement fondée sur le méta-modèle de OWL DL décrivant sa syntaxe et sa sémantique. La figure 2 illustre une représentation UML du méta-modèle de OWL DL. Elle étend les modélisations définies dans (Klein, 2004, p83) et (OMG, 2009, Chap 11).

OWL DL décrit un domaine sous forme de classes, de propriétés et d'individus. Les classes modélisent des classes d'individus. Elles peuvent être primitives ou définies. Les propriétés modélisent des relations entre classes. La propriété de subsomption permet d'organiser les classes et les propriétés en hiérarchies. Les classes et les propriétés possèdent des descriptions structurelles, élaborées à partir d'un certain nombre de constructeurs. Une sémantique est associée à chaque description de classe et de propriété par l'intermédiaire d'une interprétation (des axiomes décrivant comment interpréter les constructeurs). Les changements opérés sur les classes et les propriétés doivent être réalisés en accord avec cette sémantique. OWL DL permet aussi d'exprimer des relations entre classes telles que l'inclusion ou l'équivalence et de définir des classes complexes en appliquant des constructeurs d'intersection, d'union, de complément, de quantification de propriétés, de restrictions numériques sur les propriétés, etc.

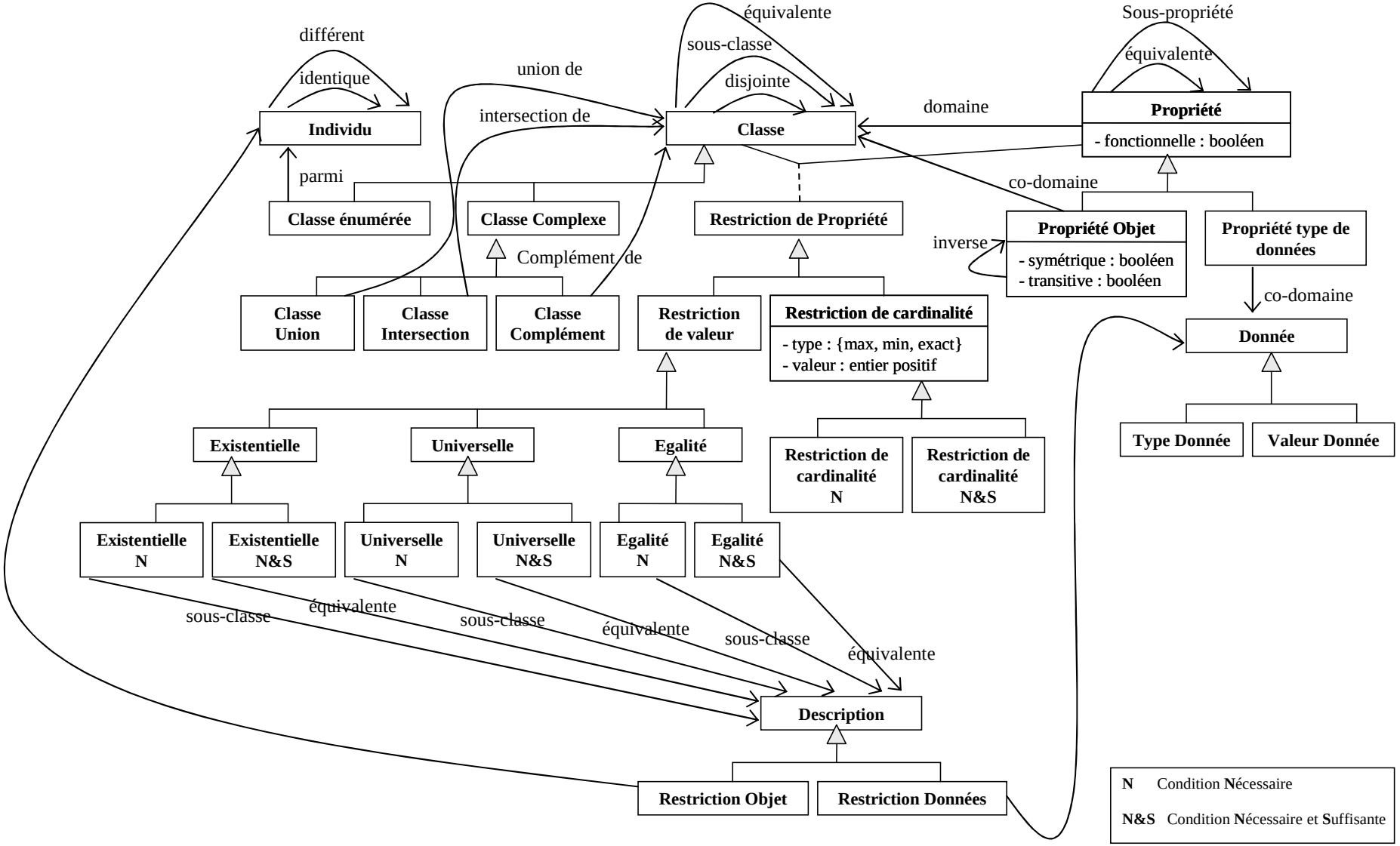


Figure 2. Représentation en UML du méta-modèle OWL.

Les changements OWL sont classés selon deux grandes catégories (Klein, 2004): des changements basiques et des changements complexes. La modélisation des patrons de changements suit cette classification.

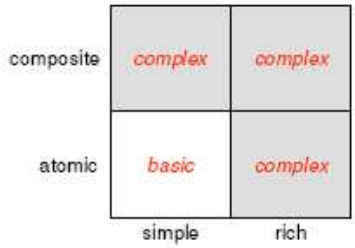


Figure 3. Classification des opérations de changements OWL (Klein, 2004, p87).

Les patrons de changements basiques correspondent à des changements simples et indivisibles qui ne modifient qu'une seule caractéristique du modèle de connaissances de l'ontologie (tels que'un changement de suppression d'une relation de subsomption). Ils modélisent tous les changements OWL DL basiques, dérivés du méta-modèle de OWL.

Les patrons de changements complexes correspondent à des changements riches incorporant des informations sur leur implication et/ou composites renfermant des séquences logiques de changements basiques et/ou complexes. Etendre le co-domaine d'une propriété à la superclasse de la classe qui le spécifiait est un exemple de changement riche. Ajouter une hiérarchie de classes (une classe et ses sous-classes) est un exemple de changement composite. Remonter les sous-classes d'une classe est un exemple de changement riche et composite.

Les patrons de changements que nous avons définis couvrent pratiquement[39] tous les changements OWL DL basiques et un premier noyau de changements complexes.

IV.1.1- Patrons de changements basiques

Les patrons de changements basiques modélisés peuvent être synthétisés par le tableau ci-après (table 6). Ils sont organisés selon le type de la primitive conceptuelle sur laquelle porte le changement (classe, propriété, ou instance).

Opérations de changements basiques		
Classes		
Classe	Ajouter, supprimer, étendre_définition[40]	
Subsomption (relation de sous-classe)	Ajouter, supprimer	
Equivalence	Ajouter, supprimer	
Disjonction	Ajouter, supprimer	
Restriction de valeur	Ajouter, Supprimer,	Type de restriction : ∀ (allValuesFrom) ∃ (someValuesFrom)

[39] Certains changements basiques de modification ne sont pas encore modélisés dans la version actuelle des patrons de changement tels que la modification d'une équivalence.

[40] C'est une modification particulière dans le sens où la définition de la classe est enrichie sans supprimer les axiomes qui la définissent déjà. Cette opération correspond à un changement basique et s'approche de l'ajout d'une classe sauf qu'elle tient compte de la définition existante et vérifie un ensemble de contraintes supplémentaires.

		∋ (hasValue)
Restriction de cardinalité	Ajouter, Supprimer	type de cardinalité : = exactly ≥ minCardinality (lowerbound) ≤ maxCardinality (Upperbound)
Propriétés		
Propriété	Ajouter, supprimer	
Domaine	Ajouter, supprimer, modifier	
Co-domaine	Ajouter, supprimer, modifier	
Subsomption	Ajouter, supprimer	
Equivalence	Ajouter, supprimer	
Fonctionelle	Mettre, annuler	
InverseFonctionelle	Mettre, annuler (si propriété objet)	
Inverse	Ajouter, supprimer, (si propriété objet)	
Symétrie	Mettre, annuler (si propriété objet)	
Transitivité	Mettre, annuler (si propriété objet)	
Instances		
Individu	Ajouter, supprimer, étendre_instanciation, supprimer_instanciation	
Identité	Ajouter, supprimer (sameAs)	
Différence	Ajouter, supprimer (differentFrom)	

Table 6. Synthèse des changements modélisés par les patrons de changements basiques.

Dans ce qui suit, nous détaillons un sous-ensemble de patrons de changements basiques, d'autres patrons de changements basiques sont présentés dans l'annexe C. Nous commençons par détailler un exemple complet de patron basique décrit au format présenté dans la section III.2 et illustré par une instanciation. Ensuite, pour le reste des patrons décrits dans cette section, nous adoptons une représentation très succincte de leurs propriétés pour ne pas alourdir le contenu du chapitre.

IV.1.1.1- Patron de changement basique d'ajout d'une relation de sous-classe

Comme nous l'avons annoncé au début de la section IV, pour expliquer comment interviennent les trois types de patrons CMP dans le guidage du processus de gestion de changement, nous illustrons par un exemple appliqué comme un fil conducteur, l'instanciation de ces patrons et des liens entre eux. A ce niveau du chapitre, nous présentons l'instanciation d'un patron de changement basique. L'exemple se base sur une ontologie minimale sur laquelle nous allons appliquer un changement basique et suivre sa gestion guidée par les patrons CMP.

Exemple : L'ontologie exemple *O* décrit les sous-classes *Animal*, *Plant*, et *Carnivorous-Plant* de la classe racine *Fauna-Flora*. Elle est définie par les axiomes suivant :

$\{Animal \sqsubseteq Fauna\text{-}Flora,$
$Plant \sqsubseteq Fauna\text{-}Flora,$
$Carnivorous\text{-}Plant \sqsubseteq Plant,$
$Plant \sqsubseteq \neg Animal\}$

Les classes *Animal* et *Plant* sont définies disjointes, c'est-à-dire que toute instance de l'une ne peut être instance de l'autre.

La description du domaine modélisé par l'ontologie *O* explique que les plantes carnivores se sont adaptées à leurs milieux de vie très pauvres et pas suffisamment nutritifs en développant des capacités à attirer, piéger et assimiler des proies. Ces proies peuvent être des insectes mais aussi des organismes tels que des invertébrés et des vers pour les plantes carnivores vivant en milieu aquatique, des petits lézards ou grenouilles pour des plantes de grande taille voire même, des petits mammifères comme des oisillons et des souris dans de rares cas[41].

Ainsi, une conceptualisation possible du monde de la faune et flore, pourrait considérer une plante carnivore comme étant une spécialisation de la classe *Plant* mais aussi de la classe *Animal* puisqu'elle adhère à certaines propriétés de cette classe. Ceci amène le besoin d'appliquer le changement *Ch1* définissant la classe *Carnivorous-Plant* comme sous-classe de la classe *Animal*.

Le patron de changement basique correspondant à l'ajout d'une relation de sous-classe et son instanciation par le changement *Ch1* sont décrit comme suit (table 7) :

EXEMPLE DE PATRON DE CHANGEMENT BASIQUE	
Propriétés générales	
Information générales	
Nom*	Ajouter une sous-classe.
Identifiant*	PChB_AjoutSous-Classe_1
Type CMP *	Patron changement.
Cas d'utilisation	
Problème	Définir une relation de sous-classe entre deux classes.
Questions de Compétence	Comment subsumer une classe par une sous-classe ? Y a-t-il des contraintes à vérifier ?
Exemples	Supposons que nous avons besoin d'exprimer que la classe *Carnivorous-Plant* est une sous-classe de la classe *Animal* dans l'ontologie *O*.
Niveau d'abstraction CMP	
Description du patron	
Intention	Le patron modélise un changement indivisible qui définit une relation de sous-classe entre deux classes et notifie s'il existe une contrainte à vérifier pour ne pas affecter la cohérence logique de l'ontologie.
Conséquences	Le patron définit une subsomption entre deux classes et notifie que la superclasse et la sous-classe ne doivent pas être disjointes.
Scénarios	Définir la classe *Carnivorous-Plant* comme une sous-classe de la classe *Animal*. Notifier que la contrainte {classe *Carnivorous-Plant* et classe *Animal* sont non disjointes} est à vérifier.
Niveau d'abstraction changement	
Description du patron	
Type patron de changement *	Patron changement basique.
Type de l'objet de changement *	Classe.
Type des entités concernées *	Classe, Classe.

[41] http://www.infoscarnivores.com

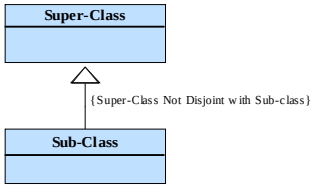
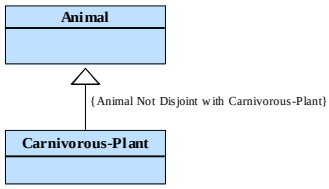
Arguments*	
Objet*	ID de la sous-classe (sub_classID).
Exemple	*Carnivorous-Plant*.
Entités référencées*	
ID Sous-Classe	ID de la sous-classe (sub_classID).
ID SuperClasse	ID de la superclasse (super_classID).
Exemples	
ID Sous-Classe	Carnivorous-Plant.
ID SuperClasse	Animal.
Contraintes	
Contraintes	¬ (sub_classID disjointWith Super_classID).
Exemples	¬ (Carnivorous-Plant disjointWith Animal).
Représentation graphique	
Diagramme	
Exemples de diagramme	
Relations	
Relations avec d'autres CMP	
Patrons incohérence	Incohérence de disjonction liée à une subsomption.

Table 7. Description du patron de changement basique « Ajouter une sous-classe ».

La modélisation du patron d'incohérences pouvant être causé par ce patron de changement est présentée dans la section IV.2.1. Celle des patrons d'alternatives pouvant le résoudre est présentée dans les sections IV.3.1 et IV.3.2.

IV.1.1.2- Patron de changements basiques d'ajout d'une classe

La modélisation du patron de changements basiques d'ajout d'une classe tient compte de la spécification de classes primitives et définies de même que des définitions intensionnelles et extensionnelles de classes.

Classe primitive – classe définie : En OWL, les classes peuvent être décrites en termes de conditions nécessaires (classes primitives) ou nécessaires et suffisantes (classes définies), contrairement aux langages de Frames qui ne supportent que les conditions nécessaires. La spécification de classes définies permet de guider les inférences de classification et de minimiser les risques d'incohérence.

Les caractéristiques nécessaires associées à une classe primitive représentent les conditions qui doivent être vérifiées si un individu est instance d'une classe (un individu *i* instance d'une classe primitive *c* possède les caractéristiques de *c*).

Les caractéristiques nécessaires et suffisantes associées à une classe définie représentent les propriétés qu'un individu doit avoir pour qu'il soit reconnu comme instance d'une classe (un individu *i* instance d'une classe définie *c*, possède les caractéristiques de *c*, et inversement, le fait qu'un individu *i* possède l'ensemble des caractéristiques associées à *c* suffit pour inférer que *i* est une instance de D).

Définition extensionnelle - définition intensionnelle : La spécification d'une classe peut être basée une *définition extensionnelle* ou une *définition intensionnelle* (Pain, 2009). La définition extensionnelle décrit la classe par l'ensemble de ses instances c'est à-dire par énumération[42] des individus membres de cette classe. Dans la définition intensionnelle, la classe est spécifiée par l'ensemble de ses propriétés.

La modélisation du patron de changements basique d'ajout d'une classe tient compte de tous ces aspects en adaptant l'implémentation du patron aux types des entités concernées et à la nature de ses arguments. Cette adaptation se matérialise conceptuellement par des spécialisations du patron.

Type des entités concernées	**Arguments**	**Contraintes**	**Axiomes OWL DL (Syntaxe abstraite)[43]**
Définir une nouvelle classe			
- Une classe	- IdClasse		`axiom ::= 'Class(' classID ')'`
Définir une collection d'union de classes			

- Une classe
- Une collection de classes
	- IdClasse
	- Id(s) Collection
	- Operateur (Union)
			`axiom ::= 'Class(' classID description1 ')'`
			Condition nécessaire et suffisante :
			`description1 ::= 'unionOf(' {description2} ')'`
			`Description2 ::= classID`

Définir une collection d'intersection de classes			

- Une classe
- Une collection de classes
	- IdClasse
	- Id(s) Collection
	- Operateur (Intersection)
		- Les classes constituant la collection ne sont pas toutes disjointes[44].	`axiom ::= 'Class(' classID description1 ')'`
			Condition nécessaire et suffisante :
			`description1 ::= 'intersectionOf('`
			`{description2} ')'`
			`Description2 ::= classID`

Définir une collection de complément de classes			

- Une classe
- Une collection de classes
	- IdClasse,
	- Id(s) Collection
	- Operateur (Complement)
			`axiom ::= 'Class(' classID description1 ')'`
			Condition nécessaire et suffisante :
			`description1 ::= 'complementOf`[45]`('`
			`{description2} ')'`
			`Description2 ::= classID`

[42] Complète ou partielle puisque c'est l'hypothèse du monde ouvert qui appliqué en OWL
[43] http://www.w3.org/TR/owl-semantics/syntax.html
[44] Autrement l'intersection sera vide. Cette contrainte permet de notifier le cas où le changement crée une classe vide.
[45] Les membres de cette classe sont définis par cette complémentarité, c'est-à-dire que tout individu qui répond à ce critère est instance de cette classe.

Définir une énumération			

- Une classe
- Une collection d'individus
	- IdClasse,
	- Id(s) Individus
			`axiom ::= 'Class(' classID description1 ')'`
			Condition nécessaire et suffisante :
			`description1 ::= 'oneOf('{individualID} ')'`

Table 8. Synthèse du patron de changements basiques d'ajout d'une classe.

IV.1.1.3- Patron de changements basiques d'ajout d'une équivalence de classes

Tout comme la définition d'un mot par synonymie, la spécification d'une classe par une équivalence de classes est une définition intensionnelle (Pain, 2009). Les classes équivalentes partagent les mêmes instances. Une équivalence peut être définie entre deux classes ou une classe et une description. Une description peut être une collection d'union de classes, d'intersection de classes, de complément de classes ou encore une énumération. C'est ce qu'on appelle des classes anonymes[46]. Elles sont définissent un ensemble d'objets du domaine mais n'ont pas de nom.

Type des entités concernées	**Arguments**	**Contraintes**	**Axiomes OWL DL (Syntaxe abstraite)**
Ajouter une équivalence d'une classe			
- Une classe - Une classe	- IdClasse - IdClasse	- Les deux classes non disjointes - Les ascendants des deux classes non disjoints	`axiom ::= 'EquivalentClasses(' classID {description1} ')'` `description1 ::= classID`
Ajouter une équivalence d'une union de classes			

- Une classe
- Une collection de classes
	- IdClasse
	- Id(s) Collection
	- Operateur (Union)
		Aucune des classes de la collection n'est (explicite-ment) disjointe de la classe ajoutée[47].	`axiom ::= 'EquivalentClasses(' classID {description1} ')'`
			`Description1 ::= 'unionOf('{description2}')'`
			`Description2 ::= classID`

Ajouter une équivalence d'une intersection de classes			

- Une classe
- Une collection de classes
	- IdClasse
	- Id(s) Collection
	- Operateur (Intersection)
		- Les classes de la collection ne sont pas disjointes entre-elles
		- Aucune des classes
			`axiom ::= 'EquivalentClasses(' classID {description1} ')'`
			`Description1 ::= 'intersectionOf('{description2}')'`
			`Description2 ::= classID`

[46] http://www.w3.org/TR/owl-guide/ et http://www.w3.org/TR/owl-ref/

[47] Par contre les classes de la collection peuvent être disjointes entre-elles. En effet, un besoin courant en construction d'ontologie est de spécifier une union de classes mutuellement disjointes telle que la classe Humain unionOf (homme, femme).

		de la collection n'est disjointe de la classe ajoutée.	
Ajouter une équivalence d'un complément de classes			

- Une classe
- Une collection de classes
	- IdClasse,
	- Id(s) Collection
	- Operateur (Complement)
		Aucune des classes de la collection n'est équivalente à la classe.	`axiom ::= 'EquivalentClasses(' classID {description1} ')'`
			`Description1 ::= 'complementOf(' {description2}')'`
			`Description2 ::= classID`

Ajouter une équivalence d'une énumération[48]			
- Une classe - Une collection d'individus	- IdClasse, - id(s) Individus	- Aucun des individus de la collection n'est disjoint de la classe. - Les instances de la classe correspon-dent uniquement aux individus spécifiés par la collection.	`axiom ::= 'EquivalentClasses(' classID {description1} ')'` `Description1 ::= 'oneOf('{individualID} ')'`

Table 9. Synthèse du patron de changements basiques d'ajout d'une équivalence de classes.

IV.1.1.4- Patron de changements basiques d'ajout d'une disjonction de classes

Une disjonction exprime qu'un individu membre d'une classe ne peut être également instance d'autres classes. Elle est définie de manière explicite ou héritée par inférence, et est mutuelle (dans les 2 sens). La sémantique des axiomes de disjonction est très profonde, particulièrement sous l'hypothèse du monde ouvert. Tant qu'il n'y a pas de disjonctions – explicites ou inférées – entre les classes de l'ontologie, celles-ci peuvent « se chevaucher » et plusieurs inférences peuvent être générées (classification, instanciation, etc.).

Une disjonction peut être définie entre deux classes ou une classe et une description. Le patron de changements basiques d'ajout d'une disjonction de classes est synthétisé par la table 10.

Type des entités concernées	**Arguments**	**Contraintes**	**Axiomes OWL DL (Syntaxe abstraite)**
Ajouter une disjonction d'une classe			
- Une classe - Une classe	- IdClasse - IdClasse	- Les deux classes non équivalentes - Pas de	`axiom ::= 'DisjointClasses(' classID description1 ')'` `description1 ::= classID`

[48] Correspond à une condition nécessaire et suffisante d'une énumération.

		descendants communs aux deux classes - Pas d'instances communes aux deux classes.	
Ajouter une disjonction d'une union de classes			
- Une classe - Une collection de classes	- IdClasse - Id(s) Collection - Operateur (Union)	Aucune des classes de la collection n'est équivalente à la classe ajoutée.	`axiom ::= 'DisjointClasses(' classID description1 ')'` **`Description1`** `::= 'unionOf(' {`**`description2`**`}')'` **`Description2`** `::=` **`classID`**
Ajouter une disjonction d'une intersection de classes			
- Une classe - Une collection de classes	- IdClasse - Id(s) Collection - Operateur (Intersection)	Les classes de la collection ne sont pas disjointes entre-elles - Aucune des classes de la collection n'est équivalente à la classe.	`axiom ::= 'DisjointClasses(' classID description1 ')'` **`Description1`** `::= 'intersectionOf(' {`**`description2`**`}')'` **`Description2`** `::=` **`classID`**
Ajouter une disjonction d'un complément de classes			
- Une classe - Une collection de classes	- IdClasse, - Id(s) Collection - Operateur (Complement)	Aucune des classes de la collection n'est disjointe à la classe.	`axiom ::= 'DisjointClasses(' classID description1 ')'` **`Description1`** `::= 'complementOf(' {`**`description2`**`}')'` **`Description2`** `::=` **`classID`**
Ajouter une disjonction d'une énumération			
- Une classe - Une collection d'individus	- IdClasse, - id(s) Individus	Aucun des individus de la collection n'est équivalent à la classe.	`axiom ::= ' DisjointClasses(' classID {`**`description1`**`} ')'` **`Description1`** `::= 'oneOf('`**`{individualID}`**`')'`

Table 10. Synthèse du patron de changements basiques d'ajout d'une disjonction de classes.

IV.1.1.5- Patron de changements basiques d'ajout d'une restriction de valeur de propriété

En OWL, les spécifications de restrictions de valeur correspondent à des définitions intensionnelles (Pain, 2009).

En OWL, on peut appliquer différentes formes syntaxiques pour modéliser un même modèle logique. En effet, d'un point de vue logiques de description, ajouter une restriction de propriété à une classe revient aussi bien à définir une relation de sous-classe entre une restriction et une classe qu'à définir une équivalence entre classes définies. Néanmoins, ces deux possibilités de définition sont sémantiquement différentes. La clause OWL `equivalentClass` exprime une condition nécessaire alors que la clause `subClassOf` (issue de RDFS) exprime une condition nécessaire et suffisante[49]. Pour distinguer les restrictions de propriétés représentant des conditions nécessaires (impliquant des classes primitives) de celles représentant des conditions nécessaires et suffisantes (impliquant des classes définies), plutôt que de proposer deux types de patrons, nous complétons les arguments du patron de restriction de valeur par un paramètre supplémentaire spécifiant le type de la condition de restriction (nécessaire ou nécessaire et suffisante). Cette idée s'inspire des propositions présentées dans les travaux de (Klein, 2004).

Il existe trois types de restrictions de valeur :

- Des restrictions universelles (∀) : pour toute instance de la classe, les valeurs des instances de la propriété spécifiée par la restriction sont toutes membres de la classe indiquée par la clause `allValuesFrom` ;
- Des restrictions existentielles (∃): au moins une des instances de propriété de la classe spécifiée par la restriction doit correspondre à un membre de la classe indiquée par la clause `someValuesFrom`. Cette restriction n'est pas rigide du fait de l'hypothèse du monde ouvert qui permet d'accepter que l'ontologie ne soit pas complète et donc peut ne pas contenir d'individus satisfaisant cette restriction ;
- Des restrictions d'égalité (∋) : elles permettent de spécifier des classes en se basant sur des valeurs particulières de propriétés. Ainsi, un individu peut être membre de la classe spécifiée par restriction dès lors qu'au moins une de ses valeurs de propriétés est égale à la ressource de la clause `hasValue`.

Type des entités concernées	**Arguments**	**Contraintes**	**Axiomes OWL DL (Syntaxe abstraite)**
Ajouter une restriction de valeur universelle pour une propriété			

- Une Classe
- Une Propriété
	- idClasse
	- idPropriété
	- Composants restriction :
	-- type (∀)
	-- valeur (type de données ou littéral ou instance de classe)
	- Condition (N ou N&S)
		- Toutes[50] les valeurs existantes déjà (instances) doivent appartenir à l'ensemble spécifié.
		- Vérifier que la classe concernée par la restriction de valeur universelle, n'a pas déjà une autre restriction de valeur universelle (existante) sur la même propriété (concernée par ce changement) et dont l'ensemble spécifiant la restriction
			Condition nécessaire
			`axiom ::= 'SubClassOf(' **classID description1** ')'`
			
			Condition nécessaire et suffisante
			`axiom ::= 'EquivalentClasses(' **classID description1** ')'`
			`----------`
			`Description1` `::=` **`restriction`**
			
			`restriction` `::=`
			`'restriction(' **datavaluedPropertyID dataRestrictionComponent** ')'`
			`|'restriction(' **individualvaluedPropertyID individualRestrictionComponent** ')'`

[49] http://www.w3.org/TR/owl-guide/

[50] En considérant l'hypothèse du monde ouvert, tant qu'il n'est pas dit explicitement que la propriété d'une classe prend comme valeurs uniquement celles spécifiées, il est supposé – notamment par le raisonneur – que la propriété de la classe peut avoir d'autres valeurs.

		est disjoint à l'ensemble spécifié par ce changement (valeur de restriction). - Vérifier que la classe concernée par la restriction de valeur universelle, n'a pas déjà une autre restriction de valeur existentielle (existante) sur la même propriété (concernée par ce changement) et dont l'ensemble spécifiant la restriction est disjoint à l'ensemble spécifié par ce changement (valeur de restriction).	**dataRestrictionComponent ::=** '**allValuesFrom**(' **dataRange** ')' **dataRange** ::= datatypeID |'rdfs:Literal' |'oneOf('{dataLiteral}')' **individualRestrictionComponent ::=** '**allValuesFrom**(' **description2** ')' **Description2** ::= **classID** **|**'oneOf('**{individualID}**')'
Ajouter une restriction de valeur existentielle pour une propriété			
- Une Classe - Une Propriété	- idClasse - idPropriété - Composants restriction : -- type (∃) -- valeur (type de données ou littéral ou instance de classe) - Condition (N ou N&S)	Vérifier que la classe concernée par la restriction de valeur existentielle, n'a pas déjà une autre restriction de valeur universelle (existante) sur la même propriété (concernée par ce changement) et dont l'ensemble spécifiant la restriction est disjoint à l'ensemble spécifié par ce changement (valeur de restriction).	Condition nécessaire axiom ::= 'SubClassOf(' **classID description1** ')' Condition nécessaire et suffisante axiom ::= 'EquivalentClasses(' **classID description1** ')' **----------** **Description1** ::= **restriction** **restriction** ::= 'restriction(' **datavaluedPropertyID dataRestrictionComponent** ')' |'restriction(' **individualvaluedPropertyID individualRestrictionComponent** ')' **dataRestrictionComponent ::=** '**someValuesFrom**(' **dataRange** ')' **dataRange** ::= datatypeID |'rdfs:Literal' |'oneOf(' {dataLiteral} ')' **individualRestrictionComponent ::=** '**someValuesFrom**(' **description2** ')' **Description2** ::= **classID** **|**'oneOf(' **{individualID}** ')'
Ajouter une restriction de valeur d'égalité pour une propriété			
- Une Classe - Une Propriété	- idClasse - idPropriété - Composants restriction : -- type (∋) -- valeur	Les valeurs existantes déjà (instances) correspondent aux valeurs spécifiées.	Condition nécessaire axiom ::= 'SubClassOf(' **classID description1** ')' Condition nécessaire et suffisante axiom ::= 'EquivalentClasses('

	(type de données ou littéral ou instance de classe) - Condition (N ou N&S)		**classID description1** ')' **----------** **Description1** ::= **restriction** **restriction** ::= 'restriction(' **datavaluedPropertyID dataRestrictionComponent** ')' |'restriction(' **individualvaluedPropertyID individualRestrictionComponent** ')' **dataRestrictionComponent ::=** '**hasvalue**(' **dataLiteral** ')' **individualRestrictionComponent ::=** '**hasvalue**(' **individualID** ')'

Table 11. Synthèse du patron de changements basiques d'ajout d'une restriction de valeur.

La modélisation des patrons d'incohérences pouvant être causés par ce patron de changement est présentée dans les sections IV.2.2 et IV.2.3.

IV.1.1.6- Patron de changements basiques d'ajout d'une restriction de cardinalité

En OWL, les spécifications de restrictions de cardinalités correspondent à des définitions intensionnelles (Pain, 2009) et permettent d'inférer des classifications. En effet, les axiomes de cardinalités spécifient que les individus ayant un nombre déterminé d'instances de propriétés peuvent être inférés comme instances des classes dont les restrictions de cardinalité sont satisfaites.

Une restriction de cardinalité maximale permet de spécifier une limite supérieure et celle d'une cardinalité minimale, une limite inférieure. La clause `cardinality` permet de préciser une valeur exacte. La spécification combinée des deux permet de délimiter un intervalle numérique. Dans (Klein, 2004), seules les restrictions minimale et maximale sont considérées. Pour définir une cardinalité exacte, deux opérations de changements sont nécessaires. Ce choix, même s'il offre une spécification de cardinalité complète et de fine granularité, n'est pas suffisant puisqu'il ne permet pas de préciser exactement le nombre d'instances qui sont en relation.

Exemple[51] : pour spécifier que chaque instance de *Vintage* a au moins une instance de *VintageYear* par la propriété *`hasVintageYear`*, il est plus explicite d'appliquer une restriction de cardinalité exacte pour la propriété indiquant que chaque instance de *Vintage* a exactement une *VintageYear* que de spécifier la *`hasVintageYear`* comme propriété fonctionnelle.

Dans cette section, nous présentons succinctement la description du patron pour l'ajout d'une restriction de cardinalité maximale.

Type des entités concernées	**Arguments**	**Contraintes**	**Axiomes OWL DL (Syntaxe abstraite)**
Ajouter une restriction de cardinalité maximale pour une propriété			
- Une Classe - Une Propriété	- idClasse - idPropriété - Composants	La propriété ne doit pas être	Condition nécessaire `axiom ::= 'SubClassOf(' **classID description1** ')'`

[51] http://www.w3.org/TR/owl-guide/

	restriction :
	-- Type cardinalité (Max)
	-- Valeur cardinalité (entier positif)
	- Condition (N ou N&S)
		transitive.	Condition nécessaire et suffisante
			`axiom ::= 'EquivalentClasses(' classID description1 ')'`
			`----------`
			`Description1 ::= restriction`
			
			`restriction ::= 'restriction(' datavaluedPropertyID dataRestrictionComponent ')'`
			`|'restriction(' individualvaluedPropertyID individualRestrictionComponent ')'`
			
			`dataRestrictionComponent ::= cardinality`
			`individualRestrictionComponent ::= cardinality`
			
			`cardinality ::= 'maxCardinality(' non-negative-integer ')'`

Table 12. Synthèse du patron de changements basiques d'ajout d'une restriction de cardinalité (partie cardinalité maximale).

IV.1.1.7- Patron de changements basiques d'ajout de domaine ou co-domaine de propriété

Deux types de propriétés sont distingués en OWL : les propriétés types de données (*Datatype property*) reliant des instances de classes à des types de données (elles sont équivalentes à des attributs de classes), et les propriétés objets (*Object property*) reliant des instances de classes entre-elles.

En OWL, les domaines et co-domaines de propriétés sont exprimés par des axiomes spécifiant les conditions nécessaires pour qu'une instance puisse être associée à une classe. Le domaine d'une propriété *p* spécifie les individus qui sont potentiellement les *sujets* (par référence à RDF) des expressions ayant pour prédicat, la propriété *p*. Le co-domaine d'une propriété *p* spécifie les individus ou les valeurs de données qui peuvent être potentiellement les *objets* (par référence à RDF) des expressions ayant pour prédicat, la propriété *p*[52].

En OWL DL, un domaine de propriété (*Domain*) peut être une classe ou une description (telle qu'une collection d'union de classes). Un co-domaine de propriété (*Range*) peut être un type de donnée ou une classe/description. Les types de données correspondent à des littéraux RDF ou aux types de données du schéma XML.

Lorsqu'une propriété possède des domaines multiples, son domaine correspond à l'intersection de tous ses domaines définis et seuls les individus appartenant à tous les domaines sont des *sujets* (par référence aux triplets RDF) potentiels. Et lorsqu'elle possède des co-domaines multiples, son co-domaine correspond à l'intersection de tous ses co-domaines définis et seuls les individus ou les valeurs de données appartenant à tous les co-domaines sont des *objets* (par référence aux triplets RDF) potentiels[53].

Dans cette section, nous présentons succinctement les patrons de changements basiques correspondant à l'ajout d'un domaine de propriété (table 13). Un domaine de propriété peut être une classe, une description (une collection d'union de classes, d'intersection de classes, de complément de classes, ou une énumération), ou encore une restriction (de valeur ou de cardinalité). On peut en effet, définir le domaine

[52] http://www.w3.org/TR/owl-guide/
[53] http://www.w3.org/TR/owl-guide/ et http://www.w3.org/TR/owl-ref/

d'une propriété comme étant une classe anonyme (il en est de même pour l'ajout d'un co-domaine de propriété). Le fait d'avoir un argument complexe ne rend pas pour autant l'opération d'ajout de domaine un changement complexe (il en est de même pour d'autres opérations de changements basiques). Le changement étant de définir un domaine pour une propriété, il correspond à une opération indivisible portant sur une seule entité ontologique (la propriété en question). La classe anonyme définissant le domaine n'engendrera pas à l'ajout d'une classe à l'ontologie.

Type des entités concernées	**Arguments**	**Contraintes**	**Axiomes OWL DL (Syntaxe abstraite)**
Ajouter un domaine de propriété sous forme de classe			

- Une Propriété
- Une Classe
	- idPropriété
	- idClasse
			`axiom ::= 'DatatypeProperty('`
			`datavaluedPropertyID` `'domain('`
			`description1``'))'`
			`|'ObjectProperty('`
			`individualvaluedPropertyID` `'domain('`
			`description1``'))'`
			`Description1` `::=` **`classID`**

Ajouter un domaine de propriété sous forme d'une union de classes			

- Une Propriété
- Une Collection de classes
	- idPropriété
	- Id(s) Collection
	- Operateur (Union)
			`axiom ::= 'DatatypeProperty('`
			`datavaluedPropertyID` `'domain('`
			`description1``'))'`
			`|'ObjectProperty('`
			`individualvaluedPropertyID` `'domain('`
			`description1``'))'`
			`Description1` `::= 'unionOf('{`**`description2`**
			`}')'`
			`Description2` `::=` **`classID`**

Ajouter un domaine de propriété sous forme d'une intersection de classes			

- Une Propriété
- Une Collection de classes
	- idPropriété
	- Id(s) Collection
	- Operateur (Intersection)
		Les classes de la collection ne sont pas toutes disjointes[54].	`axiom ::= 'DatatypeProperty('`
			`datavaluedPropertyID` `'domain('`
			`description1``'))'`
			`|'ObjectProperty('`
			`individualvaluedPropertyID` `'domain('`
			`description1``'))'`
			`Description1` `::=`
			`'intersectionOf('{`**`description2`**`}')'`
			`Description2` `::=` **`classID`**

Ajouter un domaine de propriété sous forme d'un complément de classes			

- Une Propriété
- Une Collection de classes
	- idPropriété
	- Id(s) Collection
	- Operateur (Complement)
			`axiom ::= 'DatatypeProperty('`
			`datavaluedPropertyID` `'domain('`
			`description1``'))'`
			`|'ObjectProperty('`
			`individualvaluedPropertyID` `'domain('`
			`description1``'))'`
			`Description1` `::=`
			`'complementOf('{`**`description2`**`}')'`
			`Description2` `::=` **`classID`**

[54] Cette contrainte permet de prévoir la constitution d'intersections qui génèrent des incohérences. De telles intersections sont désignées comme des Antipatterns de domaine dans http://wiki.loa-cnr.it/index.php/LoaWiki:MixedDomains.

<table>
<tr><th colspan="4">Ajouter un domaine de propriété sous forme d'une énumération</th></tr>
<tr><td>- Une Propriété
- Une Collection d'individus</td><td>- idPropriété
- id(s) Individus</td><td></td><td>axiom ::= 'DatatypeProperty('
datavaluedPropertyID 'domain('
description1'))'
|'ObjectProperty('
individualvaluedPropertyID 'domain('
description1'))'
Description1 ::= 'oneOf('{individualID}')'</td></tr>
<tr><th colspan="4">Ajouter un domaine de propriété sous forme d'une restriction de valeur</th></tr>
<tr><td>- Une Propriété
- Une Restriction de valeur</td><td>- idPropriété
- Composants restriction :
-- type
-- valeur restriction
- Condition (N ou N&S)</td><td></td><td>axiom ::= 'DatatypeProperty('
datavaluedPropertyID 'domain('
description1'))'
|'ObjectProperty('
individualvaluedPropertyID 'domain('
description1'))'
Description1 ::= Restriction(X[55])

Restriction(X) ::=
<u>Condition nécessaire</u>
axiom ::= 'SubClassOf(' X description2 ')'

<u>Condition nécessaire et suffisante</u>
axiom ::= 'EquivalentClasses(' X
description2 ')'
----
description2 ::= restriction

restriction ::= 'restriction('
datavaluedPropertyID
dataRestrictionComponent ')'
|'restriction('
individualvaluedPropertyID
individualRestrictionComponent ')'

dataRestrictionComponent ::=
'hasvalue(' dataLiteral ')'
|'someValuesFrom(' dataRange ')'
|'hasvalue(' dataLiteral ')'
dataRange ::= datatypeID
|'rdfs:Literal'
|'oneOf('{dataLiteral}')'

individualRestrictionComponent ::=
'allValuesFrom(' description2 ')'
|'someValuesFrom(' description2 ')'
|'hasvalue(' individualID ')'
Description2 ::= classID
|'oneOf(' {individualID} ')'</td></tr>
<tr><th colspan="4">Ajouter un domaine de propriété sous forme d'une restriction de cardinalité</th></tr>
<tr><td>- Une Propriété
- Une Restriction</td><td>- idPropriété
- Composants restriction :
-- Type</td><td></td><td>axiom ::= 'DatatypeProperty('
datavaluedPropertyID 'domain('
description1'))'
|'ObjectProperty('</td></tr>
</table>

[55] X désigne la classe anonyme définie par la restriction.

de cardinalité	cardinalité -- Valeur cardinalité (entier positif) - Condition (N ou N&S)		**individualvaluedPropertyID** 'domain(' **description1**'))' **Description1** ::= **Restriction(X)** **Restriction(X)** ::= Condition nécessaire axiom ::= 'SubClassOf(' **X description2** ')' Condition nécessaire et suffisante axiom ::= 'EquivalentClasses(' **X description2** ')' **----------** **description2** ::= **restriction** **restriction** ::= 'restriction(' **datavaluedPropertyID dataRestrictionComponent** ')' |'restriction(' **individualvaluedPropertyID individualRestrictionComponent** ')' **dataRestrictionComponent ::= cardinality** **individualRestrictionComponent ::= cardinality** **cardinality** ::= **'minCardinality('non-negative-integer')'** **|'maxCardinality('non-negative-integer')'** **|'cardinality('non-negative-integer')'**

Table 13. Synthèse du patron de changements basiques d'ajout d'un domaine de propriété.

IV.1.2- Patrons de changements complexes

Assez souvent, un besoin de changement d'ontologie correspond à un ensemble de changements basiques. Par exemple, lorsqu'un utilisateur souhaite ajouter une nouvelle classe et ses sous-classes, l'opération de changement dans ce cas, inclut la définition de la nouvelle sous-hiérarchie de classes et son positionnement dans l'ontologie (figure 4). Ainsi, une demande de changement peut nécessiter l'application de plusieurs changements à la fois sous forme d'une suite logique. Il est important d'appliquer ces changements en une seule transaction et de garder l'information qu'ils répondent ensemble, à un même besoin de changement. Ceci permettra de suivre l'évolution de l'ontologie, de retrouver la trace des justifications de changement et en cas d'annulation d'annuler tous les changements comme une opération de retour-arrière sur une transaction (*Rollback*) en système de gestion de bases de données. De plus, les effets d'un ensemble de changements basiques appliqués en batch, ne sont pas les mêmes si, ces mêmes changements sont appliqués séparément. Ainsi, pour guider la gestion de compositions de changements, nous avons également défini des patrons de changements complexes.

La figure 4 illustre un exemple simple de composition de changement basique de même type. Le changement global requis Ch_g consiste à ajouter la classe *Etudiant* et ses sous-classes *Thésard* et *Elève-Ingénieur*, comme une nouvelle sous-hiérarchie de la classe *Personne_Ecole*. L'application du changement Ch_g est composée de trois changements Ch_i, Ch_{ii}, et Ch_{iii} correspondant à un changement basique d'ajout d'une sous-classe :

Ch_g = {Ch_i, Ch_{ii}, et Ch_{iii}} avec
Ch_i = Ajouter sous-classe (*Etudiant*, *Personne_Ecole*)
Ch_{ii} = Ajouter sous-classe (*Thésard*, *Etudiant*)
Ch_{iii} = Ajouter sous-classe (*Elève-Ingénieur*, *Etudiant*)

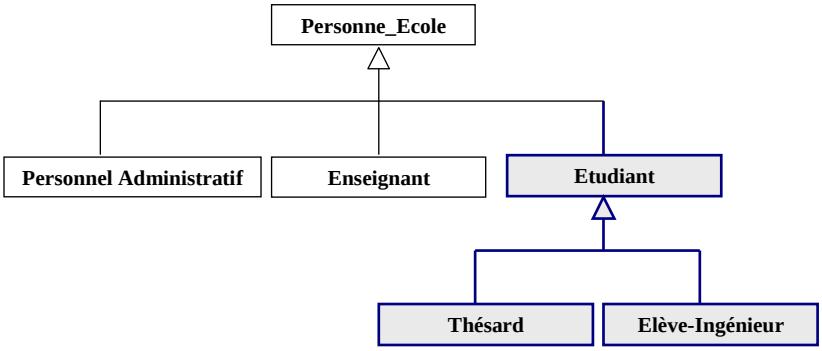


Figure 4. Exemple d'application d'une composition de changement.

Basés sur la classification des changements OWL de Klein (Klein, 2004) (figure 3), les patrons de changements complexes modélisent des compositions de changements basiques et des changements riches incorporant des connaissances supplémentaires sur le changement à appliquer ou des informations sur son implication sur le modèle logique de l'ontologie.

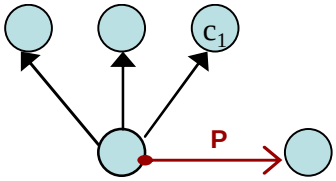


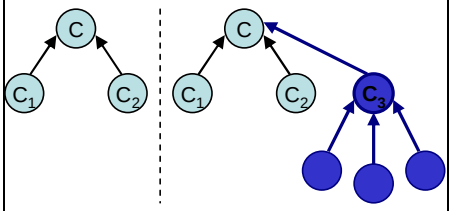
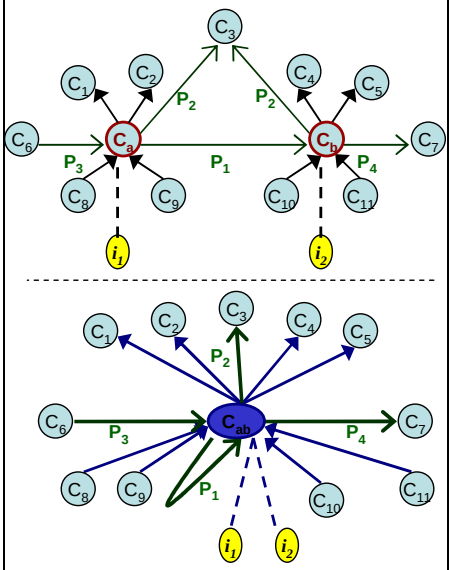
Figure 5. Exemple d'illustration pour la différence de sémantique entre changement basique et changement complexe.

Prenons un exemple simple (figure 5), une demande de changement basique « modifier domaine de la propriété *p* à c_1 » est différente d'une demande de changement complexe « étendre le domaine de la propriété p à ses superclasses ». Le changement complexe donne une information sur son impact sur le modèle logique de l'ontologie, il implique l'identification des superclasses du domaine de *p* et la définition d'autant de domaines pour *p* que ces superclasses.

Les changements complexes sont définis par des mécanismes de regroupement de changements. La modélisation de tels changements est infinie ne se basant pas sur un ensemble exhaustif de changements comme pour les changements basiques générés directement du modèle OWL DL. Il y a une infinité de possibilités de composition. Nous avons donc spécifié un premier noyau de types de changements complexes fréquemment appliqués aux ontologies et basés sur des compositions de changements sémantiquement reliés (tels que l'exemple figure 4). L'intérêt de modéliser par des patrons, des compositions de changements est d'expliquer ces compositions, de les spécifier en une suite logique ordonnée car l'imbrication des changements composants ne peut pas se faire de manière arbitraire, ceci changera la sémantique du changement global et affectera son impact sur l'ontologie. Par exemple : reclasser une sous-classe *c* d'une superclasse c_1 en la rattachant à une autre superclasse c_2 n'a pas la

même signification que de supprimer la classe *c* et d'ajouter ensuite une nouvelle classe *c* sous-classe de la classe c_2. La première composition du changement permet de préserver les instances, les propriétés et sous-classes de la classe *c*.

Les patrons de changements complexes tels que modélisés dans le modèle conceptuel des patrons CMP (figure 1) réutilisent des patrons de changements basiques et/ou complexes dans leur définition. Nous nous sommes focalisés essentiellement sur des changements complexes portant sur des classes dont voici les principaux cas[56] (table 14) :

Changement complexe	Description	Illustration
Ajouter une classe et ses sous-classes (figure 4)	Ajouter une classe et ses sous-classes comme une nouvelle sous-hiérarchie d'une classe. ⇨ Définition extensionnelle d'une classe par nomination de ses sous-classes et son positionnement dans l'ontologie Composition : 1) Ajouter une relation de sous-classe entre la classe racine de la nouvelle sous-hiérarchie et sa superclasse 2) ajouter une relation de sous-classe entre une sous-classe et la superclasse racine de la nouvelle sous-hiérarchie n) répéter 2) autant de fois que de sous-classes	
Fusionner deux classes	Fusionner deux classes en une même nouvelle classe et agréger l'ensemble de leurs propriétés, instances et sous-classes. ⇨ Fusion de deux classifications Composition (changements complexes) : 1) Ajouter une nouvelle classe fusion 2) Ajouter relation sous-classe entre la classe fusion et toutes les superclasses des deux classes 3) Ajouter relation sous-classe entre toutes les sous-classes des deux classes et la classe fusion 4) Rattacher les instances des deux classes à la classe fusion 5) Agréger les propriétés des deux classes à la classe fusion (pour toute propriété commune comme P_2, un ajustement des cardinalités sera effectué : la cardinalité min sera la minimale des deux cardinalités minimales et la cardinalité max, la maximale des deux cardinalités maximales) 6) Supprimer les deux classes fusionnées (en supprimant leurs relations avec leurs sous-classes et instances).	

[56] La liste des patrons de changements complexes est donnée dans l'annexe 2.

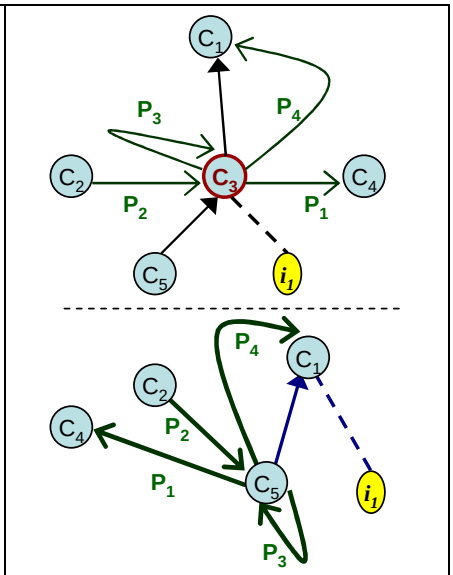
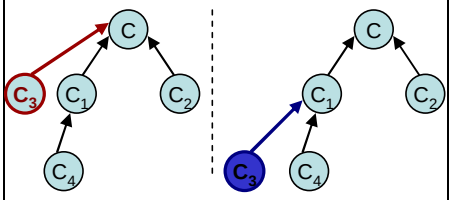
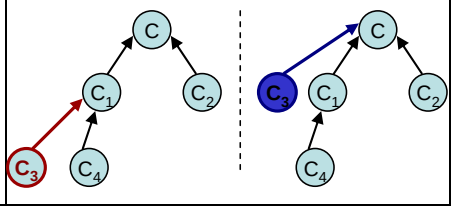
Supprimer une classe en attachant ses propriétés à ses sous-classes et ses sous-classes et instances à sa superclasse	Supprimer une classe en attachant ses propriétés à ses sous-classes et ses sous-classes et instances à sa superclasse. ⇨ Suppression d'une classe avec reclassification et redistribution de ses dépendances Composition : 1) Restreindre le domaine de toutes les propriétés dont la classe à supprimer est domaine, à ses sous-classes 2) Restreindre le co-domaine de toutes les propriétés dont la classe à supprimer est co-domaine, à ses sous-classes 3) Ajouter une relation de sous-classe entre les sous-classes de la classe à supprimer et sa superclasse 4) Supprimer l'ancienne relation de sous-classe entre la classe à supprimer et ses sous-classes 5) rattacher les instances de la classe à supprimer à sa superclasse 6) Supprimer les relations d'instanciation entres la classe à supprimer et ses instances 7) Supprimer la classe à supprimer	
Descendre une classe	Reclasser une classe en la descendant d'un niveau dans la hiérarchie : elle devient sous-classe d'une sous-classe de son ancienne superclasse. ⇨ Spécialisation d'une classification Composition : 1) supprimer l'ancienne relation de sous-classe 2) ajouter la nouvelle relation de sous-classe Rq : La sémantique de l'opération de changement est différente de celle de la généralisation d'une classification mais la composition en changements est la même. Les arguments des patrons correspondant sont par contre différents.	
Remonter une classe	Reclasser une classe en la remontant d'un niveau dans la hiérarchie : elle devient sous-classe de la superclasse de son ancienne superclasse. ⇨ Généralisation d'une classification Composition : 1) supprimer l'ancienne relation de sous-classe 2) ajouter la nouvelle relation de sous-classe	

Table 14. Description des principaux changements complexes appliqués sur les classes.

IV.1.2.1- Patron de changements complexes d'ajout d'une classe et ses sous-classes

Patron : Ajouter une classe et ses sous-classes		
Type des entités concernées		- Une classe
Arguments	Objet du changement	- IdRacine
	Entités référencées	- IdRacine -IdSuperClasse -Id(s) sous-classes de la classe racine
	Nombre sous-classes	- Nb
Contraintes		La classe racine et la superclasse non disjointes
Séquence		1) PChB_AjoutSous-Classe_1 (IdRacine, IdSuperClasse)

	2) PChB_AjoutSous-Classe_1 (IdSous-Classe n°1, IdRacine) … Nb +1) PChB_AjoutSous-Classe_1 (IdSous-Classe n°Nb, IdRacine)

Table 15. Synthèse de la description du patron de changements complexes d'ajout d'une classe et ses sous-classes.

IV.1.2.2- Patron de changement complexe de fusion de deux classes

DESCRIPTION D'UN PATRON DE CHANGEMENTS COMPLEXES	
Propriétés générales	
Information générales	
Nom*	Fusionner deux classes.
Identifiant*	PChC_Fusion_DeuxClasses_1
Type CMP *	Patron changement.
Cas d'utilisation	
Problème	Fusionner deux classes.
Questions de Compétence	Comment fusionner deux classes en une même classe ? Y a-t-il des contraintes à vérifier ?
Niveau d'abstraction CMP	
Description du patron	
Intention	Le patron modélise un changement complexe composé de changements basiques et complexes. Il fusionne deux classes et notifie s'il existe une contrainte à vérifier pour ne pas affecter la cohérence logique de l'ontologie.
Conséquences	Le patron fusionne deux classes en une même nouvelle classe et agrège l'ensemble de leurs propriétés, instances et sous-classes.
Niveau d'abstraction changement	
Description du patron	
Type patron de changement *	Patron changement complexe.
Type de l'objet de changement *	Classe.
Type des entités concernées *	Classe, Classe.
Arguments*	
Objet*	ID de la première-classe (C1_ClassID), ID de la deuxième-classe (C2_ClassID).
Entités référencées*	
ID Première-Classe	ID de la première-classe (C1_ClassID).
ID Deuxième-Classe	ID de la deuxième-classe (C2_ClassID).
ID Classe-Fusion	ID de la classe fusion (C12_ClassID)
Entités intermédiaires*	
ID(s) SuperClasses	ID(s) des superclasses directes des deux classes à fusionner. **{**super_classID**}**
ID(s) Sous-Classes	ID(s) des sous-classes des deux classes à fusionner. **{**sub_classID**}**
ID(s) Instances	ID(s) des instances des deux classes à fusionner. **{**individualID**}**

ID(s) Propriétés	ID(s) des propriétés liées aux deux classes à fusionner. **{**propertyID**}**
Nombre sous-Classes*	
Nombre sous-Classes	Nb_s-cls.
Nombre Instances*	
Nombre Instances	Nb_i.
Contraintes	
Contraintes	`Les ascendants (superclasses directes et indirectes) des deux classes à fusionner non disjointes.` `¬ (Ascendants (C1_ClassID) disjointWith Ascendants (C2_ClassID)).`
Niveau d'abstraction changement complexe	
Description du patron	
Séquence*	1) PChB_ AjoutClasse_1 (C12_ClassID) 2) PChC_AjoutSous-Classe_Multi-héritage_1 (C12_ClassID, **{**super_classID**}**) 3) Nb_s-cls fois PChB_AjoutSous-Classe_1 (sub_classID, C12_ClassID) 4) Nb_i fois PChB_EtendreInstancIndividu_1 (individualID, C12_ClassID) 5) PChC_AgrégerPropriétés_DeuxClasses_1 (**{**propertyID**}**, C1_ClassID, C2_ClassID, C12_ClassID) 6) PChB_ Supprimer_Classe_1 (C1_ClassID) 7) PChB_ Supprimer_Classe_1 (C2_ClassID)

Table 16. Synthèse de la description du patron de changement complexe « Fusionner deux classes».

La modélisation du patron d'alternatives pouvant résoudre ce patron de changement pour une incohérence de disjonction d'ascendants, est présentée dans la section IV.3.5.

IV.2- Les patrons d'incohérences

La modélisation de patrons d'incohérences permet de catégoriser des types d'incohérences logiques et de les expliciter afin de faciliter l'interprétation des résultats du raisonneur, d'orienter la localisation des axiomes responsables des incohérences et par là préparer leur résolution.

Une incohérence logique peut être causée aussi bien par un changement complexe qu'un changement basique. Les axiomes de l'ontologie doivent être vérifiés pour déceler les violations de contraintes de cohérences. Une opération de changement renseigne sur la référence des entités nommées mais n'inclut pas forcément, des informations sur le contexte d'application du changement (les axiomes de l'ontologie concernés ou pouvant être altérés, par cette opération). Il n'est donc pas possible d'anticiper intuitivement toutes les contraintes logiques liées à son application.

La modélisation des patrons CMP répond à ce problème en définissant – dans les propriétés des patrons de changements – un ensemble de contraintes à vérifier ainsi qu'un lien conceptuel entre les patrons de changements et les patrons d'incohérences permettant de prévoir les éventuels types d'incohérences qui peuvent potentiellement être causés par un type changement. Ainsi, guidée par la spécification explicite de la sémantique du changement demandé, les contraintes à contrôler d'emblée, et

l'indication des types d'incohérences attendus, l'analyse des impacts du changement se trouve orientée. Elle est complétée par l'interprétation des résultats du raisonneur et l'explication des incohérences concrètement détectées et localisées (en instanciant les patrons d'incohérences correspondants).

Les patrons d'incohérence définis modélisent un sous-ensemble d'incohérences logiques OWL DL : les incohérences de disjonction liées aux subsomptions et instanciations, les incohérences liées à la définition d'une classe sur forme d'une collection de classes, les incohérences liées aux équivalences et compléments de classes, les incohérences liées aux équivalences et disjonctions de classes, les incohérences liées aux restrictions de valeurs et les incohérences liées aux restrictions de cardinalités.

Dans ce qui suit, nous détaillons un sous-ensemble de ces patrons. D'autres patrons d'incohérences sont présentés dans l'annexe C. Nous commençons par détailler un exemple complet de patron d'incohérences de disjonction décrit au format présenté dans la section III.2 et illustré par une instanciation. Ensuite, pour le reste des patrons décrits dans cette section, nous adoptons une représentation très succincte de leurs propriétés pour ne pas alourdir le contenu du chapitre.

IV.2.1- Patron d'incohérences de disjonction liée à une subsomption

Nous distinguons deux types d'incohérences de disjonction : les incohérences de disjonction liées à des subsomptions et les incohérences de disjonction liées à des instanciations (figure 6).
Un changement d'ajout d'une relation de sous-classe entre une sous-classe et une superclasse disjointes, ou un changement d'ajout d'une disjonction entre deux classes ayant des sous-classes communes causent une incohérence de disjonction liée à la subsomption. Un changement d'ajout d'une relation d'instanciation entre une instance et une classe disjointes, ou un changement d'ajout d'une disjonction entre deux classes ayant des instances communes causent une incohérence de disjonction liée à l'instanciation.

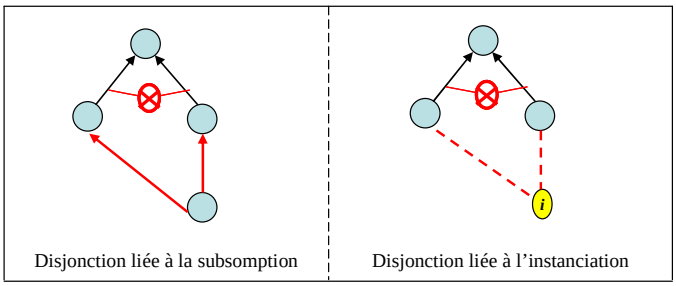


Figure 6. Illustration d'une disjonction liée à une subsomption et d'une disjonction liée à une instanciation.

Dans cette section, nous décrivons le patron d'incohérences de disjonction lié à une subsomption. Reprenons l'exemple d'application du changement *Ch1* définissant la classe *Carnivorous-Plant* comme sous-classe de la classe *Animal* (la section IV.1.1.1). Sachant que les deux classes *Animal* et *Plant* sont définies comme étant disjointes dans l'ontologie *O*, le changement *Ch1* cause une incohérence logique de disjonction liée à la subsomption entre la sous-classe *Carnivorous-Plant* et la superclasse classe *Animal*. Le patron d'incohérence correspondant et son instanciation par l'incohérence détectée suite à l'application du changement *Ch1* sont décrit comme suit (table 17) :

EXEMPLE DE PATRON D'INCOHERENCE LOGIQUE	
Propriétés générales	
Information générales	
Nom*	Incohérence de disjonction liée à une subsomption.
Identifiant*	PInc_Disjonc-Subsomp_1
Type CMP *	Patron incohérence.
Cas d'utilisation	
Problème	Analyser et délimiter une incohérence de disjonction liée à une relation de subsomption entre deux classes.
Questions de Compétence	Qu'est ce qu'une incohérence de disjonction liée à une relation de subsomption entre deux classes ? Comment l'expliquer ? Comment fournir des détails sur son analyse ?
Exemples	Supposons que nous avons besoin d'expliquer et de délimiter une incohérence de disjonction causée par la relation de subsomption entre la classe *Carnivorous-Plant* et la classe *Animal* dans l'ontologie *O*.
Niveau d'abstraction CMP	
Description du patron	
Intention	Le patron modélise explicitement l'analyse d'une incohérence de disjonction liée à une relation de subsomption entre deux classes d'une ontologie.
Conséquences	Le patron explique une incohérence de disjonction liée à une relation de subsomption et donne des détails sur son analyse et sa localisation.
Scénarios	Expliquer une incohérence de disjonction liée à la relation de subsomption entre la classe *Carnivorous-Plant* et la classe *Animal* en délimitant les classes concernées par cette incohérence et spécifiant les axiomes la causant.
Niveau d'abstraction incohérence	
Description du patron	
Arguments*	
Entités impliquées*	
ID SuperClasse1	ID d'une première superclasse (super_class1ID).
ID SuperClasse2	ID d'une deuxième superclasse (super_class2ID).
ID Sous-Classe	ID de la sous-classe (sub_classID).
Exemples	
ID SuperClasse1	Plant.
ID SuperClasse2	Animal.
ID Sous-Classe	Carnivorous-Plant
Entités concernées *	
ID SuperClasse2	ID de la superclasse concernée (super_class2ID).
ID Sous-Classe	ID de la sous-classe (sub_classID).
Exemples	
ID SuperClasse	Animal.
ID Sous-Classe	Carnivorous-Plant.
Axiomes*	
Axiomes concernés	(super_class1ID disjointWith super_class2ID), (sub_classID $\sqsubseteq$ super_class1ID).

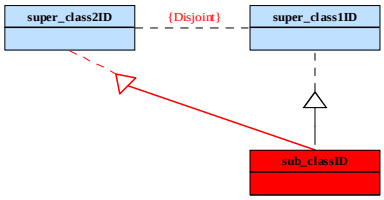
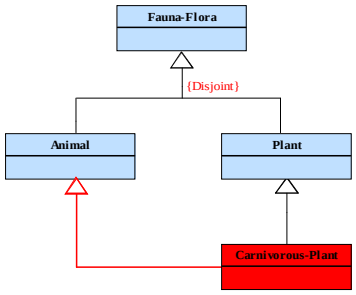
Exemples	(Plant ⊑ ¬Animal), (Carnivorous-Plant ⊑ Plant).
Axiomes responsables	(sub_classID ⊑ super_class2ID).
Exemples	(Carnivorous-Plant ⊑ Animal).
Représentation graphique	
Diagramme	
Exemples de diagramme	
Relations	
Relations avec d'autres CMP	
Patrons changement	- Ajouter une sous-classe, - Ajouter une disjonction, - …
Patrons alternative	- Définir classe hybride pour résoudre disjonction de subsomption, - Etendre définition classe pour résoudre disjonction de subsomption

Table 17. Description du patron d'incohérences « Incohérence de disjonction liée à une subsomption».

La modélisation des patrons d'alternatives pouvant résoudre ce patron d'incohérences est présentée dans les sections IV.3.1 et IV.3.2.

IV.2.2- Patron d'incohérences de restriction de valeur universelle inapplicable

Comme il a été décrit dans les contraintes du patron de changement basique « Ajouter une restriction de valeur pour une propriété » (table 11), lorsqu'il s'agit d'une restriction universelle, il faut vérifier que la classe concernée n'a pas déjà une autre restriction de valeur universelle (existante) sur la même propriété (concernée par ce changement) et dont l'ensemble spécifiant la restriction est disjoint à l'ensemble spécifié par ce changement (valeur de restriction). Cette contrainte peut être formalisée comme suit :

Pour un changement :

$C \sqsubseteq \forall P.C_2$ (C étant une classe, et P une propriété)

Vérifier que :

$\forall C_1 \in O$, si $(C \sqsubseteq \forall P.C_1)$ alors $\neg(C_1$ disjointWith $C_2)$

Cette contrainte permet ainsi d'exprimer le besoin de vérifier que la restriction universelle à ajouter ne cause pas d'incohérence logique et, qu'elle est bien applicable. Elle est matérialisée par la relation conceptuelle entre le patron de changement basique « Ajouter une restriction de valeur pour une propriété » et le patron d'incohérences « Incohérence de restriction de valeur universelle inapplicable ». Si le changement appliqué est bien un ajout d'une restriction avec type de restriction = ∀ (universelle) et qu'une telle incohérence est détectée par le raisonneur (la vérification de la contrainte donne *False*), le patron d'incohérences « Incohérence de restriction de valeur universelle inapplicable » est instancié. Une description succincte du patron illustrée d'un exemple est présentée dans la table 18.

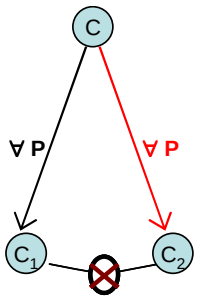
Patron : Incohérence de restriction de valeur universelle inapplicable			
Propriété	**Description**	**Exemple**	
Entités impliquées	- ClasseSourceID	C	
	- PropriétéID	P	
	- ClasseCibleDisj1ID	C_1	
	- ClasseCibleDisj2ID	C_2	
Entités concernées	- ClasseSourceID	C	
	- PropriétéID	P	
	- ClasseCibleDisj2ID	C_2	
Axiomes concernées	- (ClasseCibleDisj1ID disjointWith ClasseCibleDisj2ID)	(C_1 disjointWith C_2)	
	- ClasseSourceID ⊑ ∀PropriétéID. ClasseCibleDisj1ID	C ⊑ ∀ P. C_1	
Axiomes responsables	- ClasseSourceID ⊑ ∀PropriétéID. ClasseCibleDisj2ID	C ⊑ ∀ P. C_2	

Table 18. Synthèse de la description du patron d'incohérences « Incohérence de restriction de valeur universelle inapplicable ».

La modélisation du patron d'alternatives pouvant résoudre ce patron d'incohérences est présentée dans la section IV.3.6.

IV.2.3- Patron d'incohérences d'incompatibilité de restriction de valeur universelle et existentielle

Comme il a été décrit dans les contraintes du patron de changement basique « Ajouter une restriction de valeur pour une propriété » (table 11), lorsqu'il s'agit d'une restriction existentielle ou universelle, il faut vérifier que la classe concernée n'a pas déjà une autre restriction de valeur respectivement universelle ou existentielle (existante dans l'ontologie) sur la même propriété (concernée par ce changement) et dont l'ensemble spécifiant la restriction est disjoint à l'ensemble spécifié par ce changement (valeur de restriction). Ces deux contraintes peuvent être formalisées comme suit :

Contrainte 1 :

Pour un changement :

$C \sqsubseteq \exists P.C_2$

Vérifier que :

$\forall C_1 \in O$, si $(C \sqsubseteq \forall P.C_1)$ alors $\neg(C_1$ disjointWith $C_2)$

Contrainte 2 :

Pour un changement :

$C \sqsubseteq \forall P.C_2$

Vérifier que :

$\forall C_1 \in O$, si $(C \sqsubseteq \exists P.C_1)$ alors $\neg(C_1$ disjointWith $C_2)$

Ces deux contraintes permettent ainsi d'exprimer le besoin de vérifier que la restriction existentielle (Contrainte 1) ou universelle (Contrainte 2) à ajouter ne cause pas d'incohérence logique d'incompatibilité avec une restriction respectivement universelle ou existentielle existante. Elles sont matérialisées par la relation conceptuelle entre le patron de changement basique « Ajouter une restriction de valeur pour une propriété » et le patron d'incohérences « Incohérence d'incompatibilité de restriction de valeur universelle et existentielle». Si le changement appliqué est bien un ajout d'une restriction avec type de restriction = $\exists$ (universelle) ou $\forall$ (universelle) et qu'une telle incohérence est détectée par le raisonneur (la vérification de Contrainte1 ou Contrainte1 donne *False*), le patron d'incohérences « Incohérence d'incompatibilité de restriction de valeur universelle et existentielle » est instancié. Une description succincte du patron est présentée dans la table 19.

Patron : Incohérence d'incompatibilité de restriction de valeur universelle et existentielle		
Propriété	**Description**	**Exemple**
Entités impliquées	- ClasseSourceID	C
	- PropriétéID	P
	- ClasseCibleDisj1ID	C_1
	- ClasseCibleDisj2ID	C_2
Entités concernées	- ClasseSourceID	C
	- PropriétéID	P
	- ClasseCibleDisj2ID	C_2
Axiomes concernées	- (ClasseCibleDisj1ID disjointWith ClasseCibleDisj2ID)	(C_1 disjointWith C_2)
	Si changement restriction = $\exists$ - ClasseSourceID $\sqsubseteq \forall$PropriétéID**.** ClasseCibleDisj1ID Si changement restriction = $\forall$ - ClasseSourceID $\sqsubseteq \exists$PropriétéID**.** ClasseCibleDisj1ID	$C \sqsubseteq \forall P.C_1$ $C \sqsubseteq \forall P.C_1$
Axiomes responsables	Si changement restriction = $\exists$ - ClasseSourceID $\sqsubseteq \exists$PropriétéID**.** ClasseCibleDisj2ID Si changement restriction = $\forall$ - ClasseSourceID $\sqsubseteq \forall$PropriétéID**.** ClasseCibleDisj2ID	$C \sqsubseteq \exists P.C_2$ $C \sqsubseteq \forall P.C_2$

Table 19. Synthèse de la description du patron d'incohérences « Incohérence d'incompatibilité de restriction de valeur universelle et existentielle ».

La modélisation du patron d'alternatives pouvant résoudre ce patron d'incohérences est présentée dans la section IV.3.7.

IV.3- Les patrons d'alternatives

La modélisation de patrons d'alternatives a pour objectif de catégoriser des types d'alternatives pouvant être générés pour résoudre les types d'incohérences logiques supportés par l'approche *Onto-Evo*^{a}*l*. Ainsi, à la phase de résolution, les relations conceptuelles définies entre l'instance du patron de changement correspondant au changement requis, les patrons d'incohérences instanciés et les patrons d'alternatives

qui peuvent potentiellement les résoudre, guide la génération des instances d'alternatives de résolution pour chaque incohérence causée (figure 1).

Un patron d'alternatives modélise un type de changement à appliquer pour résoudre un type d'incohérences logiques (décrit par un patron d'incohérences) causé par un type de changement (décrit par un patron de changements). La résolution peut correspondre à un changement additionnel à appliquer conjointement au changement requis (causant l'incohérence) ou à un changement substitutif remplaçant le changement requis et dans ce cas, la sémantique du changement requis est incluse dans la spécification de l'alternative de résolution. Un patron d'alternatives est donc décrit comme un changement (basique ou complexe) et hérite des propriétés d'un patron de changements. D'autres propriétés peuvent étendre la description des patrons d'alternatives telles que les pré-conditions à satisfaire pour choisir un patron comme solution de résolution.

Dans la modélisation des alternatives pouvant résoudre les types d'incohérences spécifiés par les patrons d'incohérences définis, nous nous sommes basés sur le principe de « continuité ontologique » stipulant qu'une connaissance existante ne peut être infirmée (Xuan et al., 2006), afin de minimiser les solutions de suppression d'axiomes et d'opter plutôt pour des alternatives de division de classes, de fusion de classes, de généralisation, de redistribution d'instances, etc.

Dans cette section, nous détaillons un sous-ensemble des patrons d'alternatives définis pour les patrons d'incohérences. D'autres patrons d'alternatives sont présentés dans l'annexe C. Nous commençons par détailler deux exemples complets de patrons d'alternatives décrits au format présenté dans la section III.2 et illustré par une instanciation. Ensuite, pour le reste des patrons décrits dans cette section, nous adoptons une représentation très succincte de leurs propriétés pour ne pas alourdir le contenu du chapitre.

Reprenons encore une fois, l'exemple d'application du changement *Ch1* définissant la classe *Carnivorous-Plant* comme sous-classe de la classe *Animal* (la section IV.1.1.1). Le changement *Ch1* a causé une incohérence logique de disjonction liée à la subsomption entre la sous-classe *Carnivorous-Plant* et la superclasse classe *Animal*. Le patron d'incohérence correspondant et son instanciation ont été décrit dans la (table 17). Plusieurs alternatives peuvent être proposées pour résoudre un type d'incohérence. Pour résoudre l'incohérence décrite par le patron d'incohérences « Incohérence de disjonction liée à une subsomption », deux patrons d'alternatives peuvent être proposés : le premier correspond à une résolution substitutive étendant un changement complexe (section IV.3.1) ; le deuxième correspond à une résolution substitutive aussi mais étendant un changement basique (section IV.3.2).

IV.3.1- Patron d'alternatives de définition d'une classe hybride pour résoudre une disjonction de subsomption

EXEMPLE DE PATRON D'ALTERNATIVES	
Propriétés générales	
Information générales	
Nom*	Définir classe hybride pour résoudre disjonction de subsomption.
Identifiant*	PAL_DefClsHybResolDisjSubs_1
Type CMP *	Patron alternative.
Cas d'utilisation	
Problème	Résoudre une disjonction – liée à une subsomption – en définissant une classe hybride.
Questions de Compétence	Comment résoudre une disjonction – liée à une subsomption – en définissant une classe hybride tout en maintenant la sémantique des connaissances existantes ?

Exemples	Supposons que nous souhaitons résoudre l'incohérence de disjonction causée par la relation de subsomption entre la classe *Carnivorous-Plant* et la classe *Animal* dans l'ontologie *O*.
Niveau d'abstraction CMP	
Description du patron	
Intention	Le patron modélise une alternative résolvant une incohérence de disjonction – liée à une subsomption – en créant une classe hybride.
Conséquences	Le patron résout une incohérence de disjonction – liée à une subsomption – en définissant une classe hybride basée sur la définition des classes disjointes impliquées dans l'incohérence et en redistribuant correctement les relations de sous-classe entre les classes impliquées dans l'incohérence, la classe hybride définie et la superclasse commune la plus spécifique des classes disjointes impliquées dans l'incohérence.
Scénarios	Définir une classe hybride *Animal_Plant* basée sur la définition des deux classes disjointes impliquées dans l'incohérence *Animal* et *Plant*. Ensuite, ajouter une relation de sous-classe entre la classe hybride créée et la superclasse commune la plus spécifique des classes *Animal* et *Plant*. Enfin, substituer la relation de sous-classe entre la classe *Animal* et la classe *Carnivorous-Plant* par une relation de sous-classe entre les classes *Carnivorous-Plant* et *Animal_Plant*.
Niveau d'abstraction alternative	
Description du patron	
Processus	1) Le patron définit une classe hybride union des définitions des deux classes disjointes impliquées dans l'incohérence à résoudre ; 2) Le patron définit une subsomption entre la superclasse commune la plus spécifique des deux classes disjointes, et la classe hybride créée; 3) le patron définit une subsomption entre la classe hybride et la sous-classe concernée par l'incohérence à résoudre.
Exemples	1) Le patron définit une classe hybride *Animal_Plant* union des définitions des deux classes disjointes impliquées dans l'incohérence à résoudre *Animal* et *Plant* ; 2) Le patron définit une subsomption entre la superclasse commune la plus spécifique des deux classes disjointes *Fauna-Flora*, et la classe hybride créée *Animal_Plant* ; 3) le patron définit une subsomption entre la classe hybride *Animal_Plant* et la sous-classe concernée par l'incohérence à résoudre *Carnivorous-Plant*.
Niveau d'abstraction changement	
Description du patron	
Type patron de changement *	Patron changement complexe.
Type de l'objet de changement *	Classe.
Type des entités concernées *	Classe, Classe, Classe.
Arguments*	
Objet*	ID de la classe hybride (HybridClassID).
Exemples	Animal_Plant
Entités référencées*	
ID Sous-Classe	ID de la sous-classe (sub-ClassID)

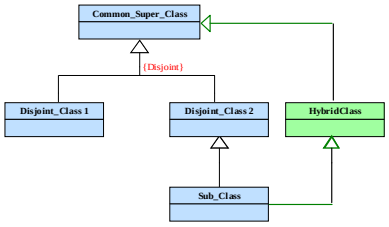
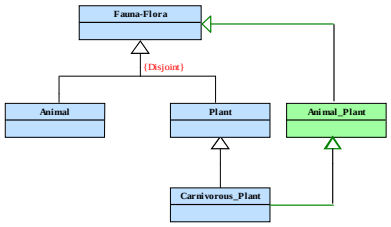
ID Première-ClasseDisj	ID de la première-classe disjointe (Disjoint_Class1ID).
ID Deuxième-ClasseDisj	ID de la deuxième-classe disjointe (Disjoint_Class2ID).
Exemples	
ID Sous-Classe	Carnivorous-Plant
ID Première-ClasseDisj	Animal.
ID Deuxième-ClasseDisj	Plant.
Entités intermédiaires*	
ID SuperClasse Commune	ID de la superclasse commune la plus spécifique des classes disjointes impliquées (Common_super_classID).
Exemples	
ID SuperClasse Commune	Fauna-Flora.
Niveau d'abstraction changement complexe	
Description du patron	
Séquence*	1) PChB_ AjoutClasse_1 (HybridClassID, Collection(Disjoint_Class1ID, Disjoint_Class2ID), Operateur(Union)) ; 2) PChB_AjoutSous-Classe_1 (HybridClassID, Common_super_classID) 3) PChB_AjoutSous-Classe_1 (sub-ClassID, HybridClassID)
Exemples	1) PChB_ AjoutClasse_1 (Animal_Plant, Collection(Animal, Plant), Operateur(Union)) ; 2) PChB_AjoutSous-Classe_1 (Animal_Plant, Fauna-Flora) 3) PChB_AjoutSous-Classe_1 (Carnivorous-Plant, Animal_Plant)
Représentation graphique	
Diagramme	
Exemples de diagramme	
Relations	
Relations avec d'autres CMP	
Patron changement	Ajouter une sous-classe.
Patrons incohérence	Incohérence de disjonction liée à une subsomption.

Table 20. Description du patron d'alternative « Définir classe hybride pour résoudre disjonction de subsomption ».

IV.3.2- Patron d'alternatives d'extension de définition d'une classe pour résoudre une disjonction de subsomption

EXEMPLE DE PATRON D'ALTERNATIVES	
Propriétés générales	
Information générales	
Nom*	Etendre définition classe pour résoudre disjonction de subsomption.
Identifiant*	PAL_EtendDefClsResolDisjSubs_1
Type CMP *	Patron alternative.
Cas d'utilisation	
Problème	Résoudre une disjonction – liée à une subsomption – en étendant la définition d'une classe.
Questions de Compétence	Comment résoudre une disjonction – liée a une subsomption – en en étendant la définition d'une classe ?
Exemples	Supposons que nous souhaitons résoudre l'incohérence de disjonction causée par la relation de subsomption entre la classe *Carnivorous-Plant* et la classe *Animal* dans l'ontologie *O*.
Niveau d'abstraction CMP	
Description du patron	
Intention	Le patron modélise une alternative résolvant une incohérence de disjonction – liée à une subsomption – en étendant la définition d'une classe.
Conséquences	Le patron résout une incohérence de disjonction – liée à une subsomption – en étendant la définition de la sous-classe concernée par l'incohérence à résoudre sur la base des définitions des classes disjointes impliquées dans l'incohérence.
Scénarios	Etendre la définition de la classe *Carnivorous-Plant* en se basant sur les définitions des classes *Animal* et *Plant*.
Niveau d'abstraction alternative	
Description du patron	
Processus	1) Le patron étend la définition de la sous-classe concernée par l'incohérence à résoudre en ajoutant – à sa description – l'union des définitions des classes disjointes impliquées dans l'incohérence.
Exemples	1) Le patron étend la définition de la sous-classe *Carnivorous-Plant* en ajoutant – à sa description – l'union des définitions des classes disjointes impliquées dans l'incohérence *Animal* et *Plant*.
Niveau d'abstraction changement	
Description du patron	
Type patron de changement *	Patron changement basique.
Type de l'objet de changement *	Classe.
Type des entités concernées *	Classe, Description Classe.
Arguments*	
Objet*	ID de la classe à étendre (ClassID).
Exemples	Carnivorous-Plant
Entités référencées*	
ID Class	ID de la classe à étendre (ClassID).

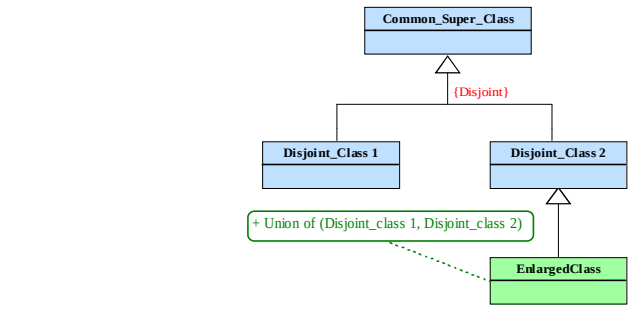
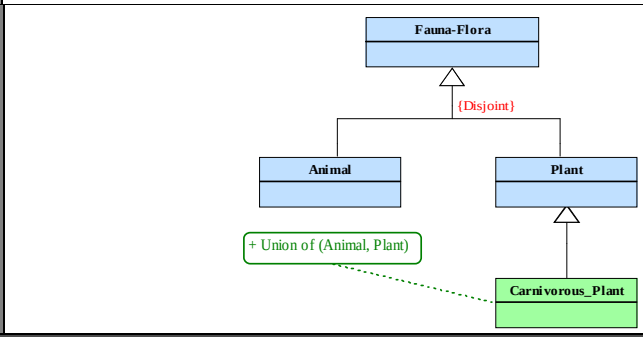
ID(s) Collection	ID(s) des classes de la collection (Disjoint_Class1ID, Disjoint_Class2ID).
Exemples	
ID Class	Carnivorous-Plant
ID(s) Collection	Animal, Plant.
Opérateur*	
Opérateur	Union
Représentation graphique	
Diagramme	
Exemples de diagramme	
Relations	
Spécialisation de	Patron de changement basique Etendre la définition d'une classe.
Relations avec d'autres CMP	
Patron changement	Ajouter une sous-classe.
Patrons incohérence	Incohérence de disjonction liée à une subsomption.

Table 21. Description du patron d'alternative « Etendre définition classe pour résoudre disjonction de subsomption ».

IV.3.3- Patrons d'alternatives de définition d'une classe hybride pour résoudre une disjonction d'instanciation

Comme mentionné au début de la section IV.2.1, deux types d'incohérences de disjonction sont distingués : les incohérences de disjonction liées à des subsomptions, et les incohérences de disjonction liées à des instances.

Une ou plusieurs instances peuvent être concernée(s) par une disjonction d'instanciation. Lorsque l'instanciation d'un individu est étendue en le rattachant à une classe qui lui est disjointe, une seule instance est concernée, l'incohérence est résolue par le patron d'alternative «définir une classe hybride pour résoudre une disjonction liée à une instance». Lorsqu'une disjonction est ajoutée entre deux classes ayant des instances communes, plusieurs instances peuvent être concernées et les instanciations aux classes disjointes doivent être supprimées. L'incohérence dans ce deuxième cas est résolue par le patron

d'alternative «Relier toutes les instances à une classe hybride pour résoudre une disjonction liée à des instances». Dans cette section, nous synthétisons la description de ces deux patrons.

Patron : définir une classe hybride pour résoudre une disjonction liée à une instance	
Processus	1) Le patron ajoute une classe hybride union des définitions de classes disjointes ; 2) Le patron définit une subsomption entre la superclasse commune la plus spécifique des classes disjointes ; 3) le patron étend l'instanciation de l'individu concerné par l'incohérernce, à la classe hybride.
Séquence	1) PChB_ Ajout_1 (HybridClassID, Collection(C1_ClassID, C2_ClassID), Operateur(Union)) ; 2) PChB_AjoutSous-Classe_1 (HybridClassID, Common_super_classID) 3) PChB_EtendreInstancIndividu_1 (individualID, HybridClassID)
Patron : Relier toutes les instances à une classe hybride pour résoudre une disjonction liée à des instances	
Processus	1) Le patron ajoute une classe hybride union des définitions des deux classes entre lesquelles est ajoutée la disjonction ; 2) Le patron définit une subsomption entre la superclasse commune la plus spécifique des deux classes entre lesquelles sera ajoutée la disjonction ; 3) Pour chaque instance commune, le patron étend l'instanciation de l'instance à la classe hybride ; 4) Pour chaque instance commune, le patron supprime le lien d'instanciation avec les classes disjointes.
Séquence	1) PChB_ Ajout_1 (HybridClassID, Collection(C1_ClassID, C2_ClassID), Operateur(Union)) ; 2) PChB_AjoutSous-Classe_1 (HybridClassID, Common_super_classID) 3) Nombre_instance fois PChB_EtendreInstancIndividu_1 (individualID, HybridClassID) 4) Nombre_instance fois PChB_ SupprimerInstancIndividu _1 (individualID, C1_ClassID) PChB_ SupprimerInstancIndividu _1 (individualID, C2_ClassID)

Table 22. Synthèse de la description des patrons d'alternatives de définition d'une classe hybride pour résoudre une disjonction d'instanciation.

IV.3.4- Patrons d'alternatives de résolution de dépendances obsolètes due à la suppression d'une classe

Le besoin de suppression d'une classe de l'ontologie peut être formulé de différentes manières. L'utilisateur peut préciser les décisions à entreprendre quant aux axiomes dépendant de la classe à supprimer (sous-classes, instances, domaines et co-domaines de propriétés). Il peut souhaiter préserver certaines connaissances et d'autres pas. Dans ce cas, le changement demandé n'est pas une simple opération de suppression de classe mais un changement complexe opérant sur plusieurs entités et regroupant une séquence de changements intermédiaires. Un exemple de patron de changement complexe de suppression d'une classe en attachant ses propriétés à ses sous-classes et ses sous-classes et instances à sa superclasse, a été décrit dans la table 14. Si par contre, aucune précision n'est donnée à la demande de suppression d'une classe, l'opération correspond à un changement basique modélisé par le patron « Supprimer une classe ». Un tel changement altère tous les axiomes référençant la classe à supprimer. Il cause une incohérence logique de « dépendances obsolètes d'une classe due à sa suppression ». Dans une telle situation, trois catégories de solutions sont envisageables : a) la suppression de toutes les dépendances obsolètes, b) le détachement de toutes les dépendances obsolètes (les rendre indépendants) ou c) la reclassification et redistribution des dépendances (table 23).

Dépendances obsolètes	Solutions envisageables	Effets
Les domaines et co-domaines des propriétés de la classe à supprimer	a) Suppression des propriétés si la classe à supprimer est leur seul domaine ou co-domaine.	Suppression de connaissances existantes.
	b) Suppression des axiomes de domaines et de co-domaines référençant la classe à supprimer.	Perte de la sémantique de la définition des domaines et co-domaines des ces propriétés.
	c) Restreindre[57] le domaine et co-domaine des propriétés référençant la classe à supprimer à ses sous-classes.	Préservation de la sémantique de la définition des domaines et co-domaines des ces propriétés.
Les sous-classes de la classe à supprimer	a) Suppression des sous-classes de la classe à supprimer (si elle est la seule superclasse).	Suppression de connaissances existantes.
	b) Suppression des relations de subsomption avec ses sous-classes (deviennent indépendantes de la classe à supprimer).	Perte de la sémantique de la spécialisation des sous-classes.
	c) Reclasser les sous-classes en les attachant aux superclasses de la classe à supprimer.	Préservation de la sémantique de la spécialisation des sous-classes.
Les instances de la classe à supprimer	a) Suppression des individus instances de la classe à supprimer (si elle est leur seule instanciation).	Suppression de connaissances existantes.
	b) Suppression des relations d'instanciations avec ses instances (deviennent indépendantes de la classe à supprimer).	Perte de la sémantique de la classification des instances.
	c) Redistribuer les instances en les attachant aux superclasses de la classe à supprimer.	Préservation de la sémantique de la classification des instances.

Table 23. Solutions envisageables pour les dépendances obsolètes suite à la suppression d'une classe.

Le choix des catégories de solutions pour les différents types de dépendances obsolètes ne se fait pas arbitrairement car certaines combinaisons sont – d'un point de vue logique – incompatibles (telles que choisir de supprimer les sous-classes et de restreindre les domaines et co-domaines aux sous-classes).

Notre choix étant de minimiser les solutions altérant les connaissances existantes (principe de continuité ontologique), nous avons renoncé aux solutions de suppression (a) et opté pour les solutions de détachement (b) et de reclassification et redistribution (c) des dépendances comme alternatives de résolution. Le choix final entre l'une ou l'autre des deux alternatives sera guidé par l'évaluation de l'impact sur la qualité de l'ontologie en tenant compte des critères de qualité et de leur pondération par l'utilisateur (chapitre 6).

Dans ce qui suit, nous décrivons succinctement les propriétés du patron d'alternatives de reclassification et redistribution des dépendances obsolètes (c), correspondant à une spécialisation d'un patron de changement complexe.

[57] La possibilité d'étendre les domaines et co-domaines des propriétés aux superclasses de la classe à supprimer, n'est pas envisagée car nous considérons que si une propriété est liée à une classe et non à sa superclasse, c'est qu'elle fait partie de la sémantique de la spécialisation de cette classe. Cette opération n'est dons appliquée que si elle est explicitement demandée par l'utilisateur.

Patron : Reclasser et redistribuer des dépendances obsolètes due à la suppression d'une classe		
Processus		1) Restreindre le domaine des propriétés référençant la classe à supprimer à ses sous-classes, 2) Restreindre le co-domaine des propriétés référençant la classe à supprimer à ses sous-classes, 3) Reclasser les sous-classes en les attachant aux superclasses de la classe à supprimer, 4) Redistribuer les instances en les attachant aux superclasses de la classe à supprimer.
Arguments	Objet du changement	- Id(s) des propriétés référençant la classe supprimée comme domaine, - Id(s) des propriétés référençant la classe supprimée comme co-domaine, - Id(s) des sous-classes de la classe supprimée, - Id(s) des instances de la classe supprimée.
	Entités référencées	- IdClasseSupprimée - Id(s) des propriétés référençant la classe supprimée comme domaine, - Id(s) des propriétés référençant la classe supprimée comme co-domaine, - Id(s) des sous-classes de la classe supprimée, - Id(s) des instances de la classe supprimée, -Id(s) des superclasses de la classe supprimée.
	Nombre propriétés (domaine)	Nb_d
	Nombre propriétés (co-domaine)	Nb_co-d
	Nombre sous-classes	Nb_s-cls
	Nombre instances	Nb-i
	Nombre superclasses	Nb_sup-cls
Séquence		1) Nb_d fois PChC_Etendre_DomProp_à_SuperClasses_1 (IdProp, IdClasseSupprimée, {Id(s)SuperClasses}) 2) Nb_co-d fois PChC_Etendre_coDomProp_à_SuperClasses_1 (IdProp, IdClasseSupprimée, {Id(s)SuperClasses}) 3) Nb_s-cls fois PChC_AjoutSous-Classe_Multi-héritage_1 (IdSous-Classe, {Id(s)SuperClasses}) 4) Nb-i fois PChC_ EtendreInstancIndividu _Multiple_1 (IdInstance, {Id(s)SuperClasses})

Table 24. Synthèse de la description du patron d'alternatives de reclassification et redistribution des dépendances obsolètes due à la suppression d'une classe.

Notons que nous ne gérons pas les axiomes liés aux collections d'intersection, d'union et de compléments de classes, contenant la classe à supprimer. Par défaut et tel que géré par l'éditeur Protégé, toutes les restrictions de valeurs et les définitions de classes basées sur ces collections sont annulées sans supprimer les classes et les propriétés qu'elles référencent (elles deviennent moins spécifiées), seule la classe à supprimer est effacée.

IV.3.5- Patrons d'alternatives de résolution de disjonction d'ascendants due à la fusion de deux classes

A l'application d'un changement complexe de fusion de deux classes (table 16), une incohérence logique peut être causée si les ascendants (les superclasses directes ou indirectes) des classes à fusionner sont disjoints. Une disjonction directe entre les deux classes à fusionner n'est par contre pas source d'incohérence puisque la suppression de cette disjonction fait partie de la sémantique du changement (la fusion).

La résolution d'un changement complexe tient compte des incohérences causées par l'application en batch de tous les changements qui le composent. L'ensemble des incohérences causées par un changement complexe sont différents de l'ensemble des incohérences relatives aux changements qui le composent. Il en est de même pour les alternatives de résolutions.

Deux alternatives sont proposées pour la résolution d'une disjonction d'ascendants due à la fusion de deux classes. Une illustration graphique simplifiée des deux patrons correspondant, est présentée par la figure 7.

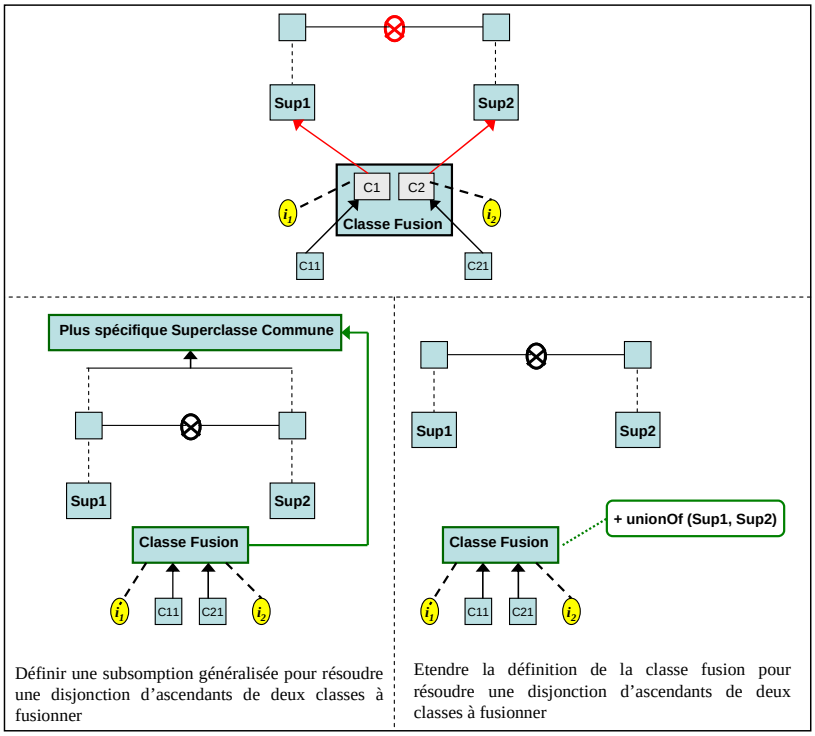


Figure 7. Illustration graphique des patrons de résolution de disjonction d'ascendants due à la fusion de deux classes.

IV.3.6- Patron d'alternatives de résolution de restriction de valeur universelle inapplicable

Ce patron modélise une résolution possible pour le patron d'incohérence de restriction de valeur universelle inapplicable décrit dans la table 18. Le rappel des axiomes liés à l'incohérence (axiomes concernés et axiomes responsables) et les transformations apportées par le patron d'alternatives sur ces axiomes sont décrits dans la table 25.

Rappel des axiomes liés à l'incohérence de restriction de valeur universelle inapplicable
$\underline{Ax_1}$: ClasseSourceID $\sqsubseteq$ $\forall$PropriétéID**.** ClasseCibleDisj1ID $\underline{Ax_2}$:ClasseSourceID $\sqsubseteq$ $\forall$PropriétéID**.** ClasseCibleDisj2ID $\underline{Ax_3}$: (ClasseCibleDisj1ID disjointWith ClasseCibleDisj2ID)
Application du patron d'alternatives de résolution de restriction de valeur universelle inapplicable
$\Rightarrow$ Transformation : supprimer Ax_1, garder Ax_3 et ajouter Ax $\underline{Ax}$:ClasseSourceID $\sqsubseteq$ $\forall$PropriétéID**.** (ClasseCibleDisj1ID $\cup$ ClasseCibleDisj2ID) $\underline{Ax_3}$: (ClasseCibleDisj1ID disjointWith ClasseCibleDisj2ID) * l'axiome Ax_2 correspondant au changement ne sera pas appliqué car la résolution définie pour ce changement est substitutive.

Table 25. Synthèse de la résolution de restriction de valeur universelle inapplicable.

IV.3.7- Patrons d'alternatives de résolution d'incompatibilité de restriction de valeur universelle-existentielle / existentielle-universelle

Ces patrons modélisent deux résolutions possible pour le patron d'incohérence d'incompatibilité de restriction de valeur universelle et existentielle décrit dans la table 19. Le rappel des axiomes liés à l'incohérence (axiomes concernés et axiomes responsables) et les transformations apportées par le patron d'alternatives sur ces axiomes sont décrits dans la table 26.

Rappel des axiomes liés à l'incohérence d'incompatibilité de restriction de valeur universelle et existentielle
$\underline{Ax_1}$: ClasseSourceID $\sqsubseteq$ $\exists$PropriétéID**.** ClasseCibleDisj1ID $\underline{Ax_2}$: ClasseSourceID $\sqsubseteq$ $\forall$PropriétéID**.** ClasseCibleDisj2ID $\underline{Ax_3}$: (ClasseCibleDisj1ID disjointWith ClasseCibleDisj2ID) Cas 1 : Changement restriction = $\exists$: les axiomes concernés sont Ax_3 et Ax_2 ; l'axiome responsable est Ax_1. Cas 1 : Changement restriction = $\forall$: les axiomes concernés sont Ax_3 et Ax_1 ; l'axiome responsable est Ax_2.
Application du patron d'alternatives de résolution d'incompatibilité de restriction de valeur existentielle – universelle (cas 1)
$\Rightarrow$ Transformation : supprimer Ax_2, garder Ax_3 et ajouter Ax $\underline{Ax_3}$: (ClasseCibleDisj1ID disjointWith ClasseCibleDisj2ID) $\underline{Ax}$:ClasseSourceID $\sqsubseteq$ $\forall$PropriétéID**.** (ClasseCibleDisj1ID $\cup$ ClasseCibleDisj2ID) * l'axiome Ax_1 correspondant au changement, dans ce cas 1, sera gardé par une résolution définie additionnelle
Application du patron d'alternatives de résolution d'incompatibilité de restriction de valeur universelle-existentielle (cas 2)
$\Rightarrow$ Transformation : garder Ax_1 et Ax_3, et ajouter Ax $\underline{Ax_1}$: ClasseSourceID $\sqsubseteq$ $\exists$PropriétéID**.** ClasseCibleDisj1ID $\underline{Ax_3}$: (ClasseCibleDisj1ID disjointWith ClasseCibleDisj2ID) $\underline{Ax}$:ClasseSourceID $\sqsubseteq$ $\forall$PropriétéID**.** (ClasseCibleDisj1ID $\cup$ ClasseCibleDisj2ID) * l'axiome Ax_2 correspondant au changement ne sera pas appliqué car la résolution définie pour ce cas 2 est substitutive.

Table 26. Synthèse de la résolution d'incompatibilité de restriction de valeur universelle et existentielle.

V- ONTOLOGIE CMP

Modéliser les patrons CMP en une ontologie permet d'attribuer à leur spécification une sémantique formelle basée sur des primitives OWL, de restreindre leur usage à travers des restrictions exprimées en logiques de description et d'assurer ainsi un certain contrôle sur leur utilisation.

Une ontologie de domaine est par définition une spécification formelle d'une conceptualisation partagée d'un domaine. L'ontologie CMP est une spécification formelle du domaine de gestion de changement d'ontologie OWL. Elle offre une modélisation explicite de changements OWL DL, d'un ensemble d'incohérences logiques que ces changements peuvent causer, et d'un ensemble d'alternatives de résolution pouvant les résoudre. C'est une sorte de « méta-ontologie » de gestion de changement d'ontologie de domaine exprimée en OWL DL, qui sert de guidage pour l'ensemble des phases du processus d'évolution et facilite son automatisation, et qui sert aussi de base pour le maintien d'un journal d'évolution**.**

La dimension « partagée » a été assurée en premier, au niveau de la méthodologie de construction de l'ontologie CMP elle-même, en se basant sur le modèle de OWL DL pour identifier des types de changements, incohérences et alternatives. Et en deuxième étape, elle sera confirmée par la mise à disposition et la diffusion des patrons CMP pour un usage public aussi bien directement, à travers le catalogue des patrons CMP sous le format de présentation décrit dans la section III.2, qu'indirectement à travers l'application du processus d'évolution *Onto-Evo*^{a}*l*.

V.1- Méthode de construction de l'ontologie de CMP

La construction de l'ontologie CMP s'est axée sur le modèle formel de OWL DL. Nous avons commencé par définir les différents types de changements basiques pouvant être appliqués à une ontologie OWL DL en se basant sur les spécifications du langage OWL DL et sur la classification des changements définie par (Klein, 2004). Toujours sur la base des spécifications de OWL DL, nous avons identifié les contraintes d'application de ces changements en se basant sur leur sémantique. L'application au cas par cas de ces changements sur des ontologies test a permis d'identifier les arguments nécessaires à leur application et de définir ainsi, les propriétés de description des patrons de changements basiques. De ces patrons, et en considérant les opérations de compositions de changements les plus fréquentes ou interdépendantes, nous avons déterminé un premier ensemble de changements complexes pour lesquels nous avons pareillement, identifié des contraintes d'application, les types d'arguments nécessaires, des propriétés de description dont certaines communes aux changements basiques, et d'autres spécifiques aux compositions, ce qui a permis de finaliser la hiérarchie de classification des patrons de changements.

A partir des contraintes d'application identifiées pour certains patrons de changements, ainsi que l'application d'instances de ces patrons sur des ontologies test pour analyser leurs conséquences sur la cohérence de l'ontologie, nous avons défini un ensemble de patrons d'incohérences logiques pouvant être causées par ces types de changements. En considérant ce lien de causalité, les résultats du raisonneur à la détection de ces types d'incohérences identifiés, et en précisant le contexte dans lequel ils se sont produits (nous distinguons par exemple, une disjonction liée à une subsomption, d'une disjonction liée à une instanciation), nous avons identifié un ensemble de propriétés expliquant ces incohérences, notamment en précisant les entités et les axiomes pouvant être impliqués dans ces incohérences et en indiquant – pour certains types d'incohérences – les axiomes responsables afin de guider leur localisation.

En tenant compte de la sémantique du changement à l'origine d'une incohérence (la raison du changement), et des dépendances entre les entités et les axiomes concernés, nous avons dérivé un ensemble d'alternatives possibles pour les types d'incohérences modélisés. Les résolutions qu'elles apportent ont été vérifiées sur des cas test. En liant une alternative à l'incohérence qu'elle résout et au

changement qui l'a causée, deux types de résolutions ont été distingués selon que l'alternative doit être appliquée conjointement au changement (résolution additionnelle) ou en remplacement en y intégrant sa sémantique dans l'alternative (résolution substitutive).

V.2- Modélisation de l'ontologie CMP

Tout comme pour la présentation de l'ontologie journal d'évolution (chapitre 3, section V.2), afin de simplifier la présentation graphique de l'ontologie CMP, nous avons choisi d'en représenter quelques parties en diagramme UML (figures 8, 11, 12, 13, 14, 15, 16) en nous basons notamment sur les travaux présentés dans (Brockmans et al., 2004). Afin de faciliter la lisibilité des figures, une codification par couleur a aussi été adoptée distinguant les différents types de patrons (en bleu, les patrons de changements, en rouge les patrons d'incohérences, et en vert les patrons d'alternatives).

Ainsi, les hiérarchies de classes de l'ontologie sont représentées par des relations d'héritage entre classes, les propriétés objets par des associations entre classes en indiquant par le sens de navigabilité, le domaine et le co-domaine de la propriété, les propriétés type de données par des attributs, les restrictions de valeurs par des contraintes sur les associations correspondantes indiquant les valeurs de restrictions, et les restrictions de cardinalités par l'expression des cardinalités des associations.

Certaines classes particulières correspondant à des relations entre plusieurs classes telles que la classe *Résolution* liant les classes *Patron Changement*, *Patron Incohérence* et *Patron Alternative* sont représentées en UML par des relations N-aires. La modélisation de ces classes en OWL s'est basée sur les recommandations[58] proposées par le groupe « Semantic Web Best Practices and Deployment Working Group » du «W3C OEP task force », et est illustrée par la figure 9.

Les compositions de classes spécifiées par la propriété objet « composée de », telles que pour les classes correspondant à des patrons de changements complexes, sont représentées par une relation d'agrégation. La séquence donnant l'ordre des compositions est modélisée par deux propriétés objets particulières *en_premier* et *suivant*[59] indiquant respectivement pour chaque composition ordonnée (telle que la séquence des changements composant un changement complexe), le premier élément de la séquence et l'élément suivant à un élément de la séquence. Chaque séquence de composition ne peut avoir qu'un seul premier élément. Cette modélisation se base sur les patrons de modélisation de séquences en OWL proposés dans (Drummond et al., 2006) pour la modélisation de séquences de protéines en ontologies biologiques. Elle est illustrée par la figure 10.

La figure 8 illustre la couche abstraite (les premiers niveaux) de l'ontologie CMP. La classe *Entité* est spécialisée par les classes *Propriété*, *Classe*, et *Individu* du méta-modèle OWL DL (figure 2). La figure 11 illustre la hiérarchie des patrons de changements basiques. La figure 12 illustre la description d'un exemple de patron de changements basiques. La figure 13 illustre la description d'un exemple de composition de patron de changements complexes. La figure 14 illustre la description d'un exemple de patron d'incohérences avec ses spécialisations et les patrons de changements qui leurs sont liés. La figure 15 montre comment le type (responsable ou concerné) d'un même axiome d'un patron d'incohérences dépend du changement qui a causé l'incohérence et illustre la description d'un axiome. La figure 16 illustre les résolutions correspondantes à un patron de changements pour chacun des patrons d'incohérences qu'il peut causer. D'autres illustrations sont présentées dans l'annexe D. La description de l'ontologie en OWL DL est présentée à l'annexe E.

[58] http://www.w3.org/TR/swbp-n-aryRelations/

[59] Nous avons recours également à la propriété *suivant** lorsque qu'un composant s'enchaine plus d'une fois.

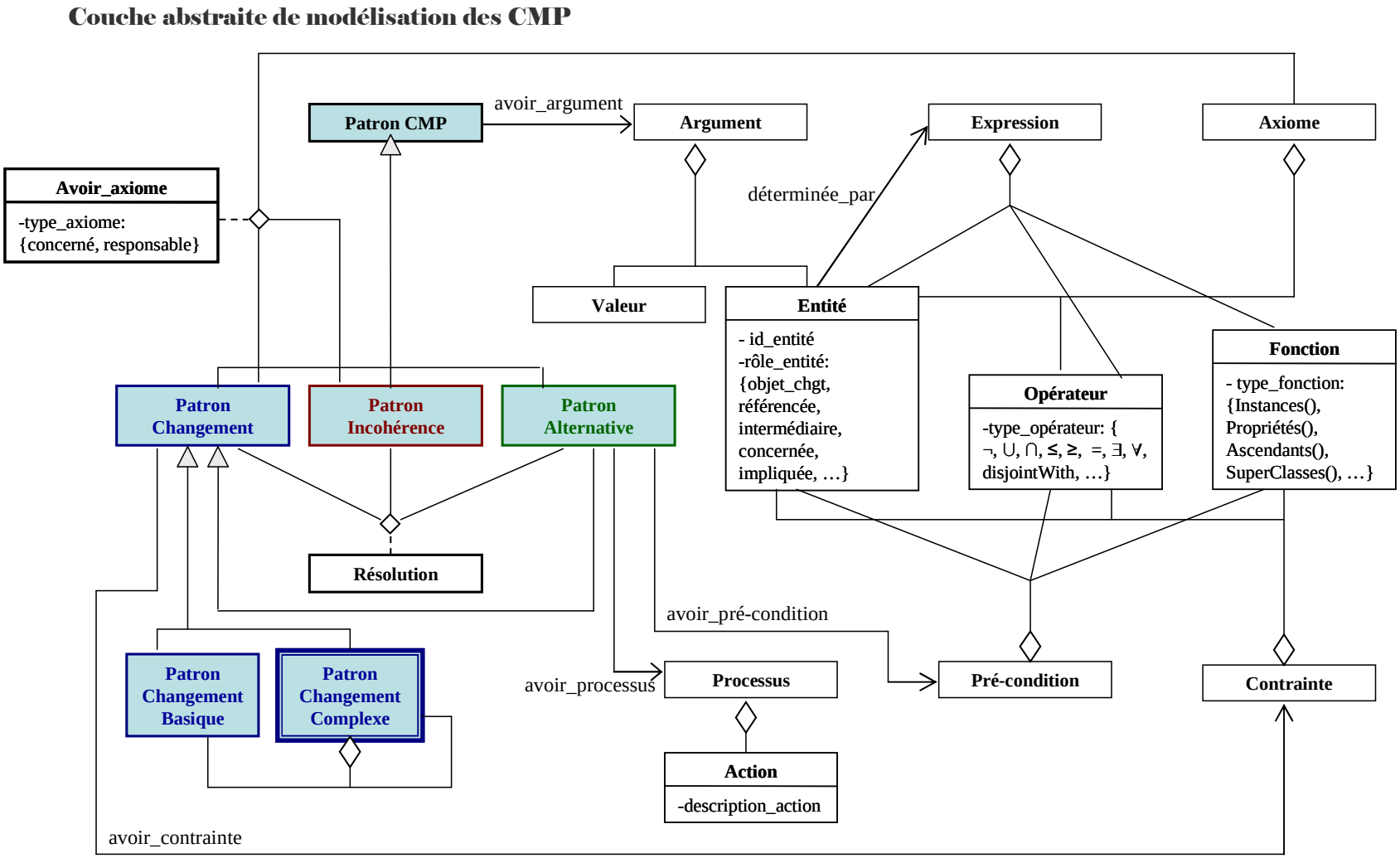


Figure 8. Représentation graphique en UML de la couche abstraite de l'ontologie CMP.

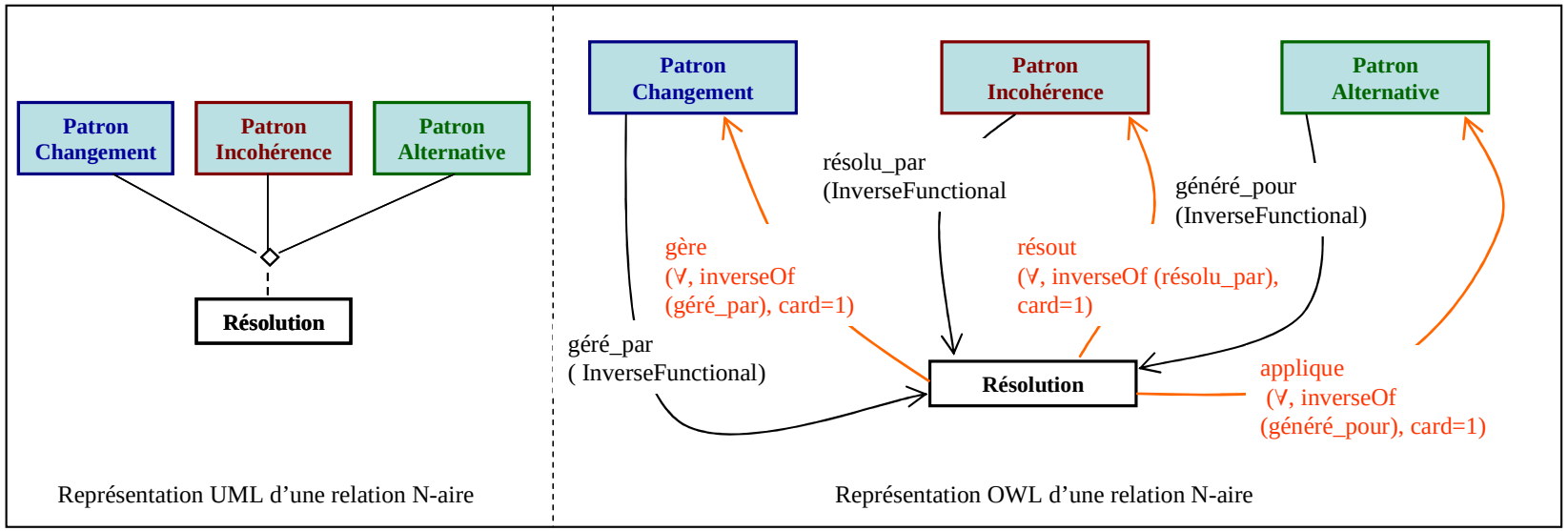


Figure 9. Représentation en UML et en OWL d'une relation N-aire.

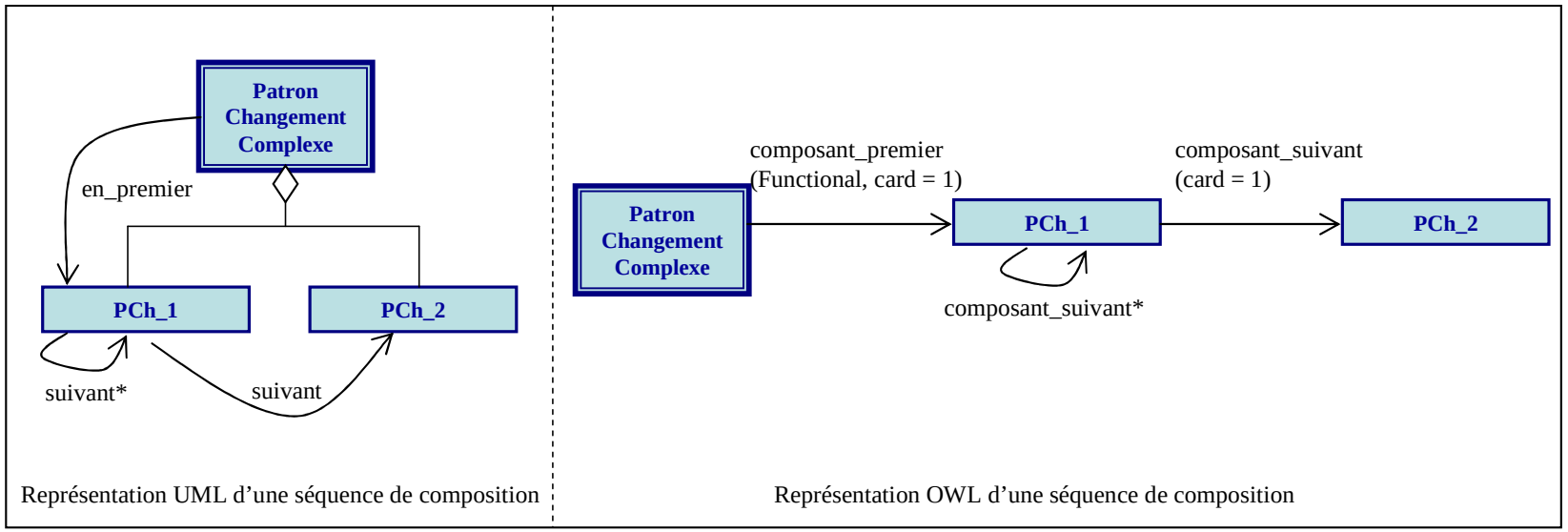


Figure 10. Représentation en UML et en OWL d'une séquence de composition.

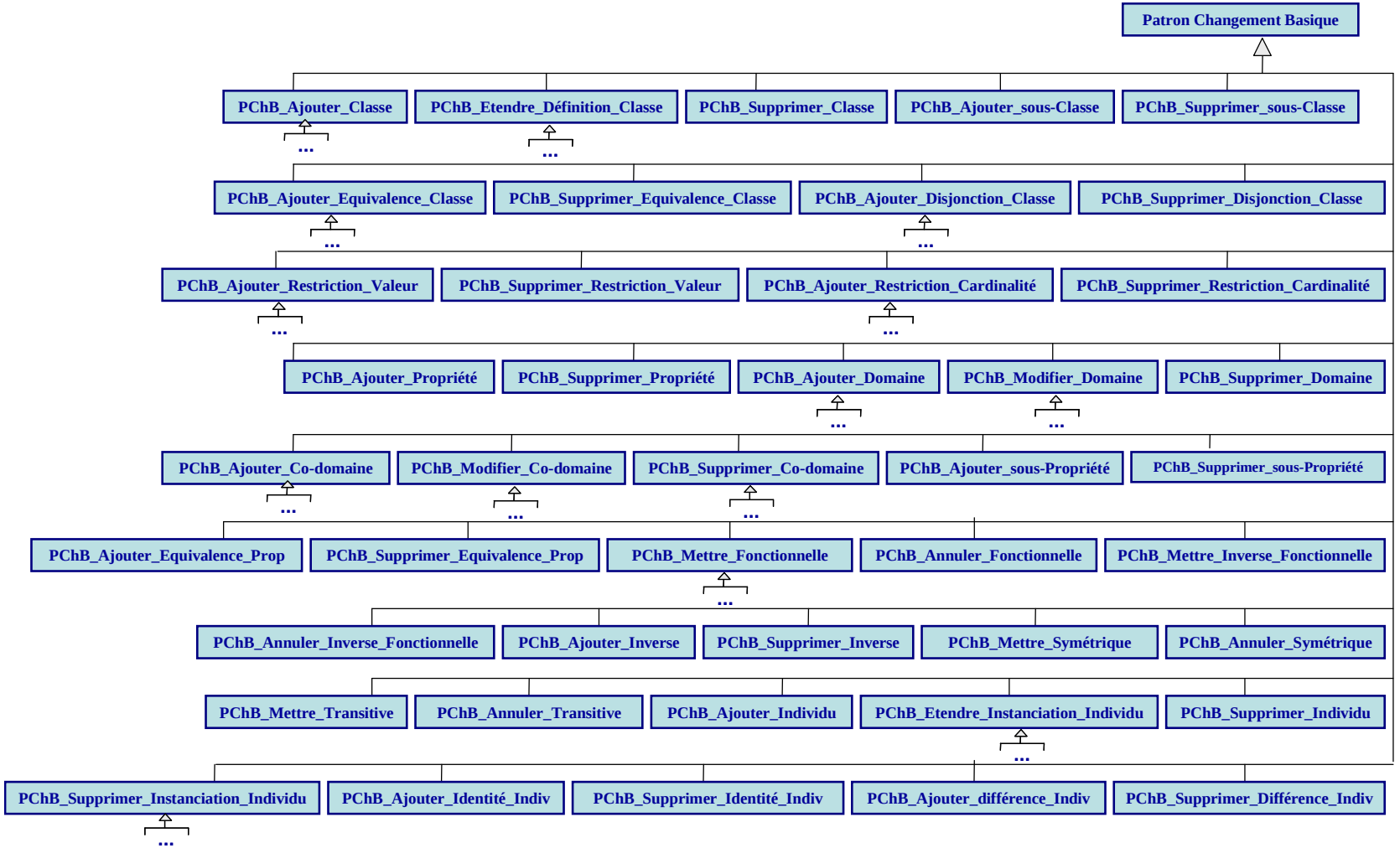


Figure 11. Représentation graphique en UML de la hiérarchie des patrons de changements basiques.

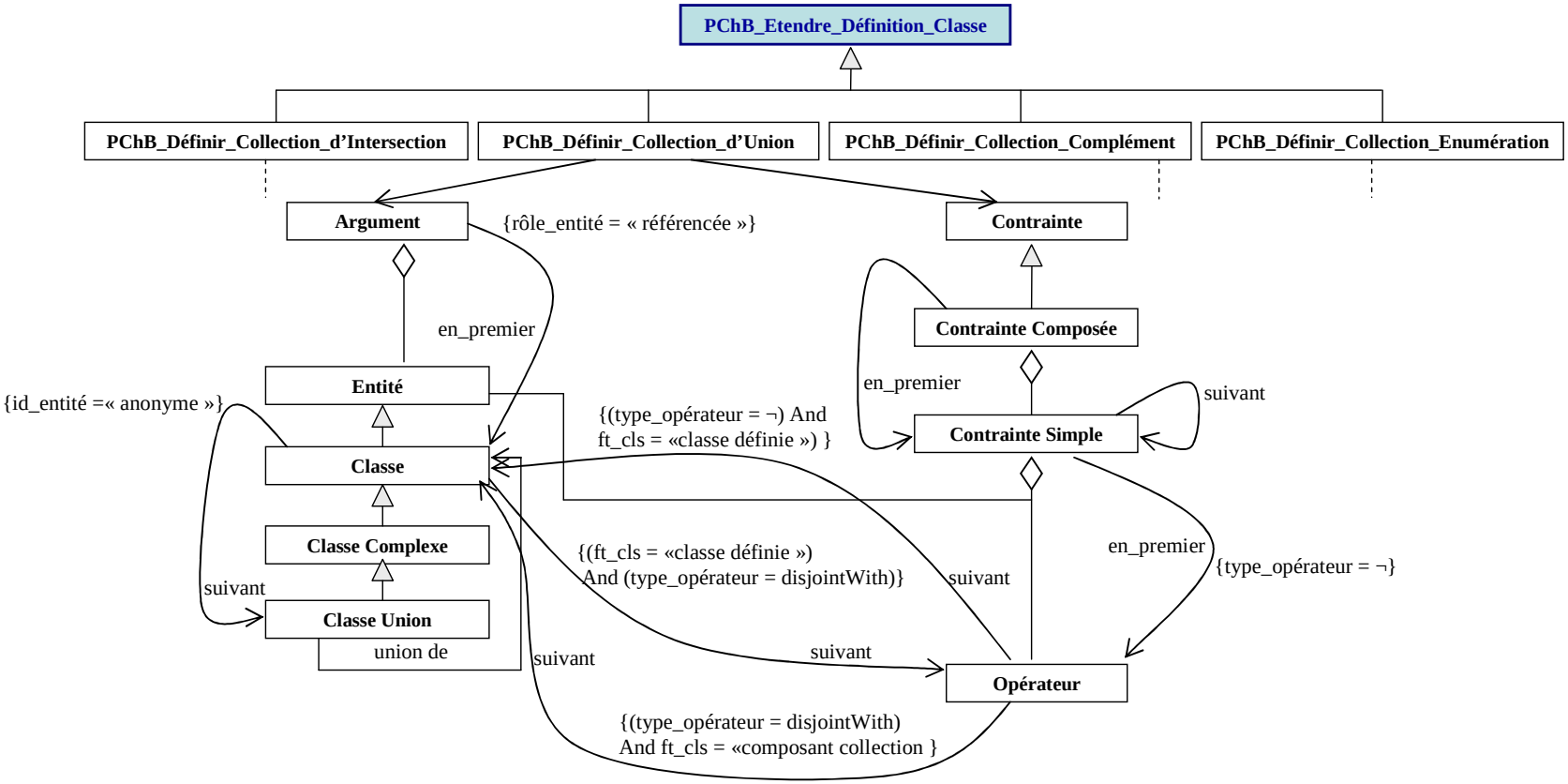


Figure 12. Représentation graphique en UML de la modélisation d'un exemple de patron de changement basique « Etendre la définition d'une classe ».

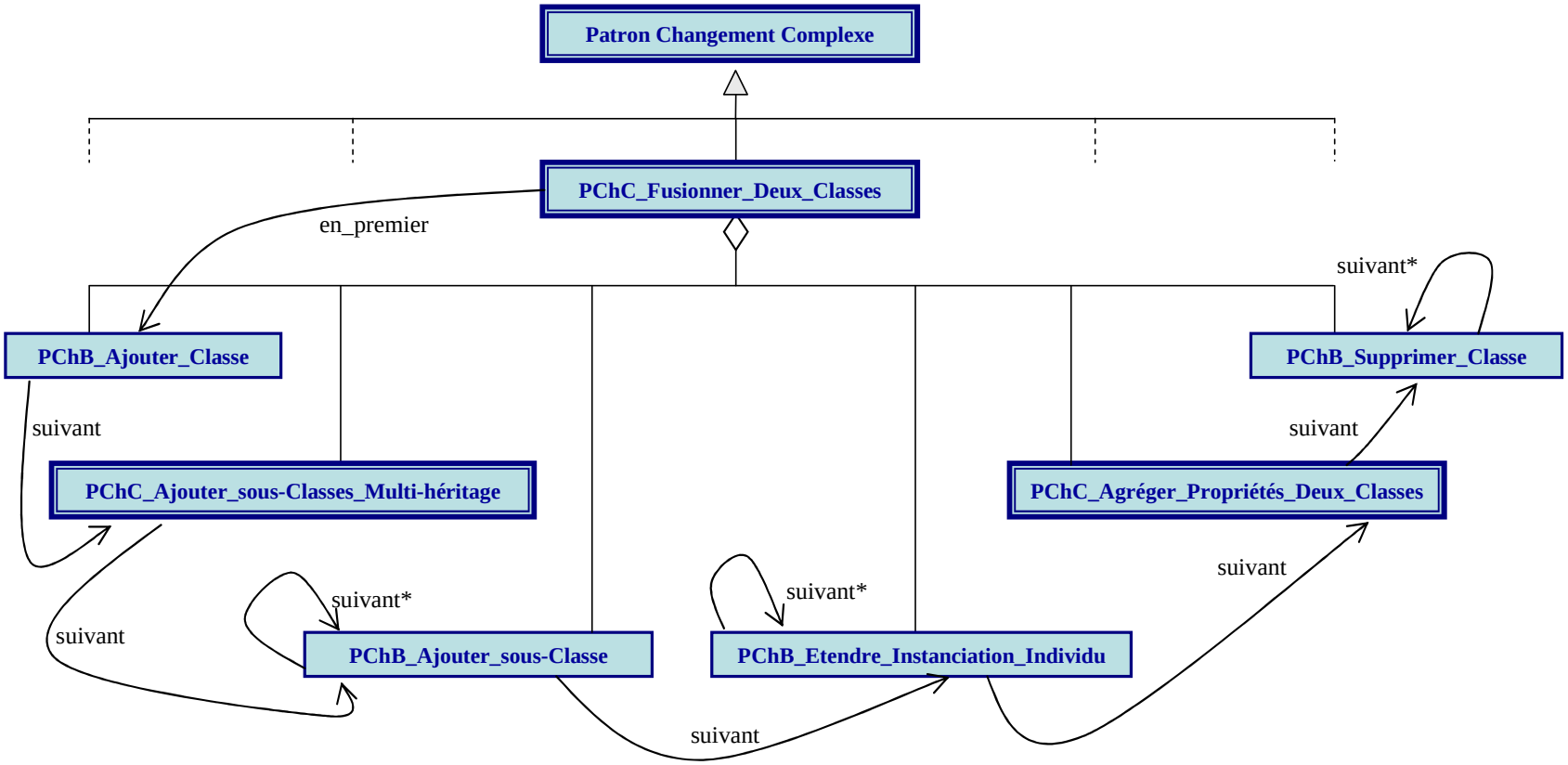


Figure 13. Représentation graphique en UML de la modélisation d'un exemple de patron de changement complexe « Fusionner deux classes ».

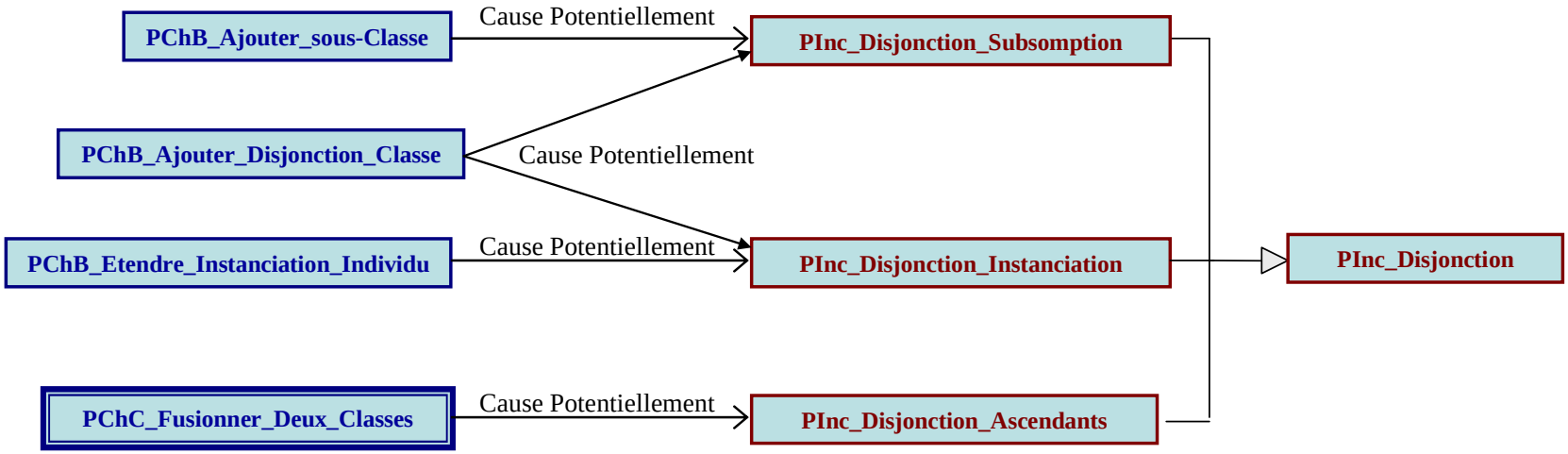


Figure 14. Représentation graphique en UML de la modélisation d'un exemple de patron de d'incohérence de disjonction avec ses spécialisations et les patrons de changements qui leurs sont liés.

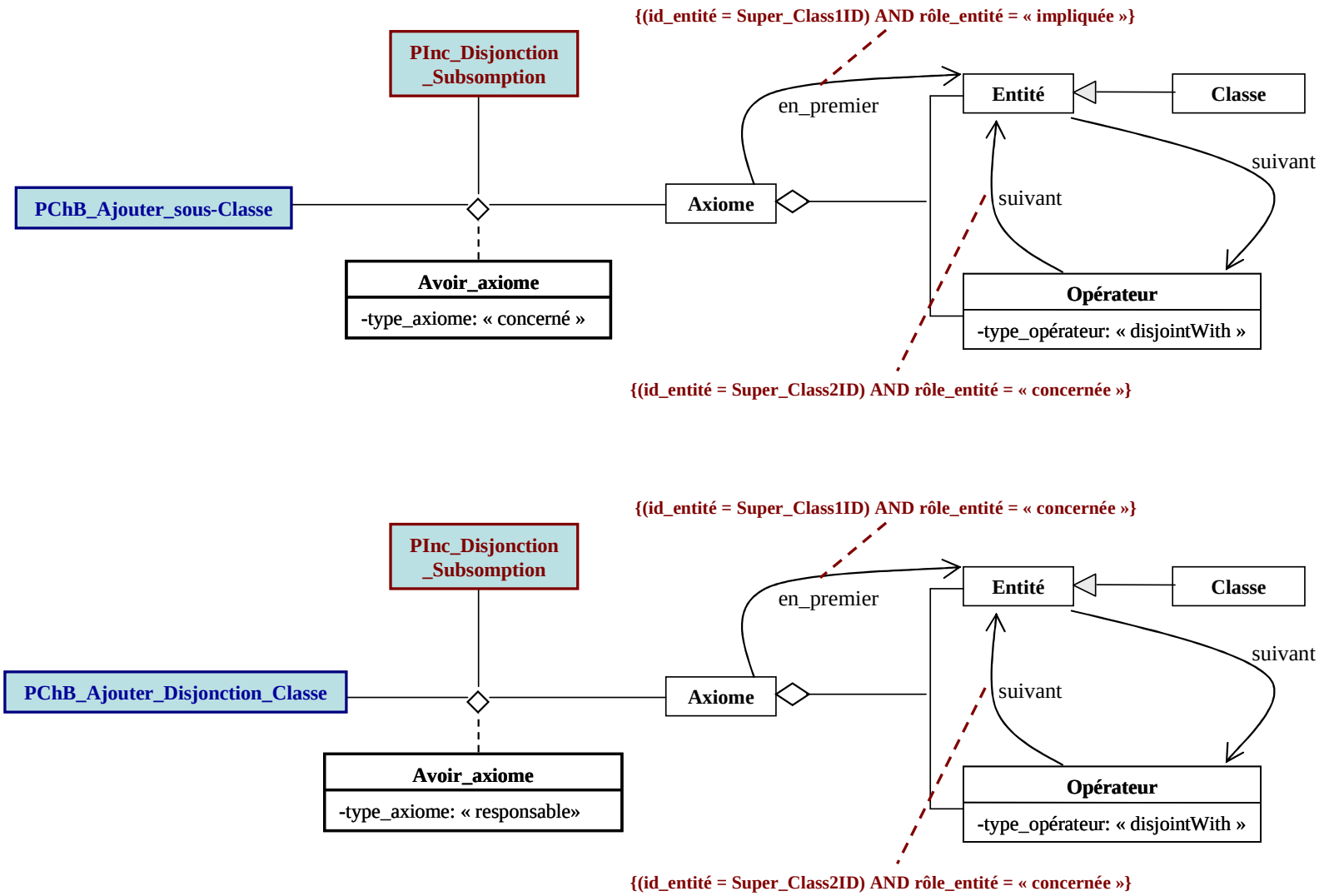


Figure 15. Représentation graphique en UML de la modélisation de la dépendance du type des axiomes des patrons d'incohérences aux changements qui les ont causés et illustration de la description d'un axiome.

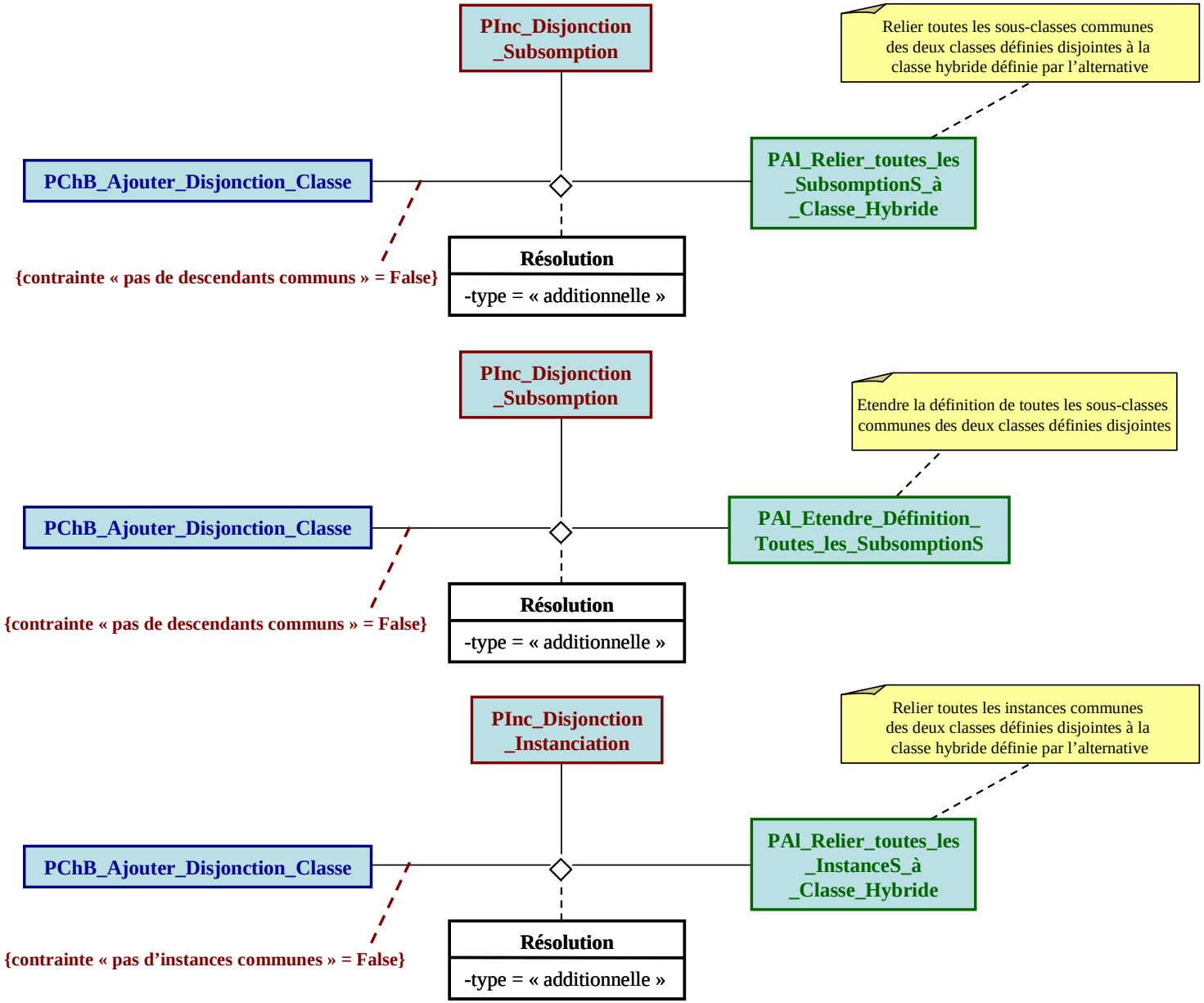


Figure 16. Représentation graphique en UML de la modélisation des résolutions du patron de changement « Ajouter disjonction de classe » pour chacun des patrons d'incohérences qu'il peut causer : « Disjonction de subsomption » et « disjonction d'instanciation ».

VI- DISCUSSION

Après la présentation des patrons de gestion de changement CMP et la description de l'ontologie CMP, nous synthétisons dans cette section, les choix de représentation des patrons CMP, les principales caractéristiques de leur modélisation, et l'approche déclarative de gestion de changement qu'ils offrent. De plus, nous discutons de quelques observations diverses liées à leur modélisation et nous nous positionnons par rapport aux travaux existants.

VI.1- Représentation des patrons CMP

Les patrons CMP sont représentés selon deux couches :

- Une couche de présentation sous forme d'un catalogue de patrons CMP décrits en langage naturel et illustrés par des diagrammes UML. Elle facilite l'échange, le partage et la documentation des patrons CMP notamment pour des besoins d'évaluation de ces patrons ;
- Une couche de spécification formelle sous forme d'une ontologie OWL DL des patrons CMP définissant la sémantique des patrons (à travers des subsomptions, des compositions, des restrictions de propriétés), des relations entre eux, et de leur application. Elle facilite le guidage du processus de gestion de changement.

La modélisation en ontologie des CMP et l'organisation des différents patrons en hiérarchies allant du plus générique au plus spécifique permet de bénéficier des principes d'héritage et de spécifier des propriétés communes à plusieurs patrons (arguments, relations sémantiques avec d'autres patrons, des restrictions, etc.).

La représentation formelle des CMP est encore plus utile lorsqu'elle est définie dans le même modèle que l'ontologie de domaine. L'ontologie CMP étant exprimée en OWL DL, facilite outre le guidage du processus de gestion de changement, le stockage des connaissances sur les patrons CMP et aussi la réutilisation de composants de manipulation d'ontologie dans l'implémentation de l'approche *Onto-Evo*[a]*l.*

VI.2- Les caractéristiques de modélisation des patrons CMP

La modélisation des patrons CMP a pour objectif de définir des traitements de gestion de changement typiques relativement à des classes de changements en associant à un type de changement, les contraintes logiques à vérifier, les incohérences susceptibles d'être causées et les alternatives permettant de les résoudre (les changements dérivés nécessaires pour maintenir la cohérence). Elle facilite l'automatisation de la gestion des changements notamment:

- En spécifiant des contraintes de granularité fine pour faciliter l'instanciation et l'interprétation des patrons ;
- En tenant compte de la variabilité des types des arguments (une classe simple, une collection de classes ou d'individus, des restrictions, des classes anonymes, etc.) et de la portée des axiomes (axiomes appliqués sur toute une classe ou sur une restriction de classe, etc.) ;
- En distinguant les axiomes explicites (liés aux entités référencées directement par le patron) des axiomes implicites (liés aux entités intermédiaires manipulées par le patron).

La modélisation des patrons CMP permet également de comprendre, partager et réutiliser des modèles de changements d'ontologie OWL DL. Par ailleurs, la définition de patrons de changements complexes facilite la gestion de ces changements et la rend plus efficace puisque le détail des changements composants et leur ordonnancement sont explicitement spécifiés tout en préservant la sémantique du changement complexe dans sa globalité. En effet, pour des spécifications de changement sémantiquement

différentes, il est possible d'avoir en résultat – une fois ces spécifications explicitées – un même modèle logique de changement c'est-à-dire, des compositions de changement équivalentes. Un changement de fusion d'une sous-classe avec sa superclasse par exemple, équivaut à un changement de suppression de la sous-classe en rattachant toutes ses propriétés, instances et sous-classes à sa superclasse.

Ainsi, à la demande d'application d'un changement, l'utilisation des patrons CMP permet de :

- Définir la signification du changement requis (sa spécification explicite, sa portée, etc.) ;
- Le formaliser et l'associer à un patron de changements ce qui constitue sa signature ;
- Déduire – en tenant compte de la structure de l'ontologie et des contraintes du modèle OWL DL – ses impacts attendus (éventuels) sur la cohérence et ce, à partir :
  - du patron de changements instancié : les contraintes à vérifier pour que le changement soit applicable tout en maintenant la cohérence,
  - de la relation conceptuelle entre le patron de changements en question et les patrons d'incohérences qui lui sont liés. Cette relation renseigne sur les incohérences que ce type de changement peut potentiellement causer.
- Détecter ses impacts réels sur la cohérence en se basant sur les schémas d'axiomes de l'ontologie initiale (contraintes logiques exprimées dans l'ontologie). La détection des incohérences réellement causées suite à l'application du changement, fait intervenir un raisonneur. Les résultats d'analyse et les notifications d'incohérences par le raisonneur, sont formatés et complétés pour instancier les incohérences détectées aux patrons d'incohérences correspondants.
- Proposer à partir des patrons d'incohérences instanciés, les patrons d'alternatives pouvant les résoudre en se basant sur la classe *résolution* liant les instances des trois types de patrons et aussi sur la structure de l'ontologie en évolution et les contraintes logiques qui y sont exprimées.

VI.3- Patrons CMP pour une approche déclarative de gestion de changements

Les patrons CMP permettent de bénéficier d'une approche déclarative de gestion de changements plutôt qu'une approche appliquant des méthodes procédurales de résolution d'incohérences (Muetzelfeldt, 2006). Les approches procédurales décomposent le changement en sous-parties et traitent chaque partie par une procédure qui analyse et résout son impact sur la cohérence (Stojanovic et al., 2002a). Elles ne permettent pas (ou peu) de réutiliser des invariances de traitement comme des modèles. De même, la justification des traitements n'est pas évidente car le traitement interne est difficile à comprendre. Dans l'approche *Onto-Evo^{a}l*, la maintenance de la cohérence pour l'application d'un changement, se base sur un ensemble complet, prédéfinis de patrons (traduits par des axiomes) formalisant la gestion de ce type de changement. Les résolutions générées sont vérifiées par rapport à la cohérence puis, triées en considérant des critères de qualité d'ontologie. Tout comme les approches déclaratives, *Onto-Evo^{a}l* assure la spécification de la structure conceptuelle et de traitement du système d'évolution c'est-à-dire, des objets et des arguments manipulés par le processus de gestion de changements (descriptions des patrons CMP), de même que les relations fonctionnelles entre eux (relations entre les différentes patrons CMP).

VI.4- Observations diverses

1ère observation

Les deux propriétés objet *composant_premier* et *composant_suivant* de l'ontologie CMP (représentées en UML par les relations UML *en_premier* et *suivant*) jouent un rôle très important dans le guidage du processus d'évolution. Elles permettent de déployer des séquences de compositions (listes ordonnées de composants). La propriété *composant_premier* indique le premier élément d'une séquence de

composition. La propriété *composant_suivant* indique l'élément (action, axiome, contrainte, alternative, changement composant) suivant à générer ou à vérifier. En spécifiant l'ordre d'application des changements composants et celui de la vérification des contraintes, elles guident par exemple, l'analyse et la résolution d'un changement complexe.

Outre l'instanciation de ces deux propriétés du niveau processus de l'approche *Onto-Evo*^{a}*l*, au niveau historique à travers les classes *Trace_changement_complexe*, *Trace_changement_basique*, *Trace_incohérence*, *Trace_alternative* et *Trace_résolution* de l'ontologie journal d'évolution (Chapitre 3, section V.2.1); d'autres propriétés de précédence (telles que *trace_résolution_précédante*, *trace_évolution_précédente*, *version_entité_précédante*, etc.) viennent les compléter pour indiquer la dernière opération générée en traçant l'évènement précédant d'un évènement enregistré. Chaque nouvelle trace est ainsi liée à la dernière trace – de même type – enregistrée. Ces propriétés de précédence jouent un rôle très important pour la traçabilité de l'application réelle d'un changement en permettant de retrouver l'ordre des séquences de traitements de changements et d'évolution des entités de l'ontologie et de l'ontologie elle-même.

2ème observation

Il est possible d'avoir comme argument d'un patron de changements basiques, une classe complexe anonyme (description d'une collection ou une restriction de valeur). Ainsi, un domaine de propriété peut être aussi bien une classe de l'ontologie, qu'une description de collection (correspondant à une collection d'union, d'intersection, ou de complément de classes ou encore une énumération d'individus) ou encore une restriction de valeur. Le modèle OWL DL permet en effet, de définir le domaine d'une propriété comme étant une classe complexe anonyme (il en est de même pour la définition d'un co-domaine de propriété). Le fait d'avoir un argument complexe ne rend pas pour autant l'opération d'ajout de domaine un changement complexe (il en est de même pour d'autres opérations de changements basiques ayant pour argument une classe complexe). Le changement étant de définir un domaine pour une propriété, il correspond à une opération indivisible portant sur une seule entité de l'ontologie (la propriété en question). La classe anonyme définissant le domaine n'engendrera pas à l'ajout d'une classe à l'ontologie. La modélisation de la gestion du patron de changement tient compte par contre, de la sémantique de cette classe anonyme et de son impact sur la cohérence de l'ontologie.

3ème observation

Les changements basiques d'ajout d'un domaine (ou co-domaine) défini par une intersection de classes ou les changements de modification d'un domaine (ou co-domaine) en y ajoutant une intersection de classes, impliquent de vérifier – la contrainte – que les classes constituant l'intersection ne sont pas disjointes sinon il y aura « potentiellement » génération d'incohérences. Potentiellement parce que ce n'est pas une incohérence détectée par le raisonneur dans tous les cas mais uniquement lorsqu'il y a déjà des individus de ces classes disjointes qui instancient la propriété en question. Cette *incohérence d'intersection de domaines disjoints* potentiellement causée par les changements cités ci-dessus, est modélisée et supportée par les patrons CMP même si elle n'est pas forcément détectée par les raisonneurs. Ainsi, pour une demande de ce type de changements, le processus vérifie s'il y a disjonction entre les classes de l'intersection et propose – tout en préservant la sémantique du changement – une alternative prévue pour la résoudre : *transformer domaine intersection en domaine union*. Une telle résolution anticipe de probables incohérences détectables plus tard (après une instanciation par exemple) qui peuvent être – en

dehors du contexte d'application du changement définissant le domaine – difficiles à expliquer. Ce raisonnement rejoint les propositions d'anti-patterns de domaines et leur résolution par LOA[60].

VI.5- Positionnement par rapport aux travaux existants

VI.5.1- Patrons d'incohérences et anti-patterns logiques

Dans (Corcho et al., 2009), les auteurs identifient un ensemble de patrons – communément reproduits par des experts de domaine dans la conception d'ontologies OWL – engendrant des incohérences. Ils sont appelés des anti-patterns. Plusieurs types d'anti-patterns ont été définis. Les anti-patterns logiques – représentant des erreurs que les raisonneurs basés sur les logiques de description peuvent détecter – peuvent être comparés aux patrons d'incohérences particulièrement, l'anti-pattern *OnlynessIsLoneliness* se rapprochant du patron d'incohérence de *restriction de valeur universelle inapplicable* et l'anti-pattern *UniversalExistence* se rapprochant du patron d'incohérences *d'incompatibilité de restriction de valeur universelle et existentielle.*

VI.5.2- Patrons de changements et ontologie des opérations de changements OWL

Dans (Klein, 2004), une ontologie d'opérations de changements OWL a été définie classant les changements OWL en changements basiques et changements complexes. La modélisation des patrons de changements de l'ontologie CMP s'est basée sur cette classification. Cependant, l'objectif de modélisation étant différent (guidage du processus de gestion de changement et non la classification des opérations de changements pouvant être répertoriées dans une approche de détection de changements déjà appliqués), les patrons de changements sont plus détaillés que les types de changements définis dans (Klein, 2004). Pour un changement d'ajout d'une classe par exemple, nous définissons plusieurs spécialisations de patron d'ajout d'une classe tenant compte des différentes possibilités de spécification d'une classe (une intersection de classes, un complément de classes, etc.), de leur sémantique et de leur contrainte d'application par rapport à la cohérence de l'ontologie.

Dans (Klein, 2004), les opérations de modifications ont été intégrées dans la liste des opérations de changements et distinguées des opérations de succession de suppression-ajout puisque les logs des outils d'ontologies sur lesquels se base l'approche de détection de changements contiennent des informations sur les modifications. Pareillement, nous avons modélisé des changements de modification dans les patrons de changements (étendre la définition d'une classe, modifier le domaine ou le co-domaine d'une propriété, étendre l'instanciation d'un individu). La traçabilité de ces changements est assurée au niveau du journal d'évolution à travers les classes *Version_entité* et *Ligne_de_vie_entité* (Chapitre 3, section V.2.2). De plus, ces classes permettent de sauvegarder le re-nommage d'une entité, supporté directement au niveau du journal d'évolution sans déclencher le processus de gestion de changement puisqu'il n'a pas d'impact sur la cohérence[61] de l'ontologie et nécessite juste la mise à jour des axiomes liés à l'entité (ce qui se fait automatiquement par l'éditeur d'ontologie). Dans (Klein, 2004), ces changements sont classés comme des changements de labels de ressources.

Nous avons cependant, restreint les opérations de modification prises en charge pour simplifier la gestion que suscite l'application de tels changements. Ainsi, nous n'intégrons pas par exemple, dans les

[60] http://wiki.loa-cnr.it/index.php/LoaWiki:MixedDomains

[61] Rappelons que l'approche *Onto-Evo*^{a}*l* est définie pour la gestion d'évolution d'ontologies dans un contexte local c'est-à-dire, sans tenir compte d'une dimension collaborative ou distribuée de l'ontologie. Autrement, une opération de re-nommage même si elle n'a pas d'impact sur la cohérence logique de l'ontologie évoluée, doit être propagée aux artefacts dépendants.

patrons de changements, les modifications du contenu d'une restriction de valeur ou de son type (d'universelle à existentielle ou vice versa) ou la modification du type d'une propriété (de propriété objet à propriété type de données ou vice versa). De même, pour préserver la sémantique de certains changements (notamment ceux liés aux subsomptions de classes et aux disjonctions et équivalences de classes ou d'individus) et surtout gérer leur impact sur la cohérence de l'ontologie, nous ne modélisons pas d'opérations de modification. Ces opérations sont traitées comme une succession de suppression et d'ajout.

VI.5.3- Gestion des changements complexes

La demande d'un changement complexe nécessite l'application de plusieurs changements à la fois sous forme d'une suite logique constituant une même transaction. Les effets d'un ensemble de changements basiques appliqués en batch, ne sont pas les mêmes si, ces mêmes changements sont appliqués séparément (section IV.1.2). Ainsi, un changement complexe est géré dans sa globalité, en considérant son impact global sur la cohérence et en résolvant les incohérences causées par l'application regroupée de tous les changements qui le composent. Cette gestion des changements complexes diffèrent du procédé proposée dans (Plessers et al., 2007). Dans une approche de détection de changement, les auteurs gèrent une demande de changement à travers quatre étapes : i) la maintenance de la cohérence, ii) la détection des changements intermédiaires appliqués pour résoudre l'incohérence, iii) le rétablissement de changements (Change Recovery) en supprimant les changements intermédiaires non nécessaires (par application de leur changements inverses) et iv) l'implémentation du changement final. Pour une demande de changement complexe, ces étapes sont appliquées à chacun des changements composants, leur résultats sont regroupés à la fin pour constituer l'implémentation du changement global. Certes, une optique de détection de changements appliqués est différente d'une optique de suivi et de gestion de l'application d'un changement, nous pensons néanmoins que le traitement par décomposition d'un changement complexe finit par compliquer sa gestion et altérer sa sémantique originelle.

VI.5.4- Introduction de patrons d'évolution de bases de connaissances RDF

Dans (Auer et Herre, 2007), une approche de support d'évolution d'ontologies RDF distribuées a été présentée. L'approche prend en charge des changements atomiques d'ajout ou de suppression de triplets RDF et des changements composés résultant de hiérarchies de changements définies selon une approche ascendante. Les hiérarchies ne modélisent pas des spécialisations de changements mais plutôt des niveaux de compositions. Ainsi, une hiérarchie constituée d'un concept C_1 ayant deux sous-classes C_2 et C_3 indique que l'application du changement C_1 à un graphe RDF implique l'application des changements C_2 puis C_3 en une seule transaction et que les changements C_2 et C_3 correspondent à des changements atomiques. Cependant aucune indication n'est donnée sur la spécification de l'ordre d'application des changements composants.

Les changements appliqués à un graphe RDF sont représentés dans une ontologie log sous forme d'instances d'une classe `log:Change` (Auer et Herre, 2007). Chaque changement atomique est représenté comme une réification de déclarations RDF et référencé par les propriétés `log:added` ou `log:removed` selon que le changement soit un ajout ou une suppression. Une propriété `log:parentChange` relie une instance de changement à une instance de changement composé. La séquence d'application des changements composants n'est cependant pas représentée.

Un ensemble de patrons d'évolution a été introduit pour faciliter la révision des changements appliqués dans un contexte distribué, et aussi pour étendre le Framework à des ontologies OWL. Chaque patron peut être un patron positif ou négatif d'évolution selon qu'il corresponde à un changement d'ajout

ou de suppression. Un patron est défini par un triplet (X, G(X), A(X)) avec X un ensemble de variables, G(X) un patron de graphe caractérisant un changement avec l'ensemble des variables X, et A(X) l'algorithme de migration des données permettant de propager le changement à l'environnement distribué de l'ontologie (Auer et Herre, 2007). Les variables X définissent les paramètres du changement tels que les URI des entités sur lesquelles s'applique le changement.

Afin de comparer avec la modélisation des patrons de changements, nous illustrons – en partie – un exemple de patron correspondant à l'ajout d'une cardinalité maximale (Auer et Herre, 2007). L'ensemble des variables correspondant à ce changement est X = (class, property, maxCardinality). Le patron de graphe G(X) correspondant est illustré par la figure 17.

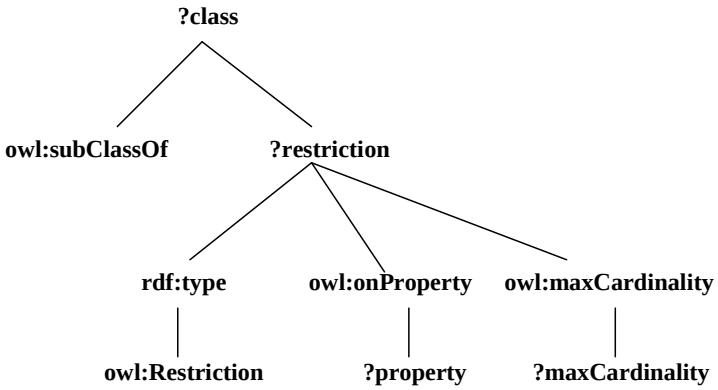


Figure 17. Illustration d'un exemple de patron d'ajout de cardinalité maximale proposé dans (Auer et Herre, 2007).

Les auteurs ont aussi introduit des possibles stratégies d'application d'un changement de suppression de classe mais sans détailler le patron d'évolution correspondant. Ces stratégies sont :

- Supprimer une classe en la fusionnant à sa superclasse (les instances, les sous-classes et les propriétés sont ramenées à la superclasse) ;
- Trancher une classe en préservant juste les propriétés (supprimer les sous-classes et les instances et rattacher les propriétés à la superclasse) ;
- Supprimer complètement une classe et sa sous-hiérarchie (supprimer les instances, les sous-classes et les propriétés qui lui sont directement liées).

VI.5.5- CMP et ODP

Une des premières perspectives à court terme est d'intégrer les patrons CMP au portail des Ontology Design Patterns ODP[62]. Ceci permettra de mettre à disposition une librairie de patrons de gestion de changement pour les partager et les évaluer. Les patrons CMP peuvent être intégrés comme un type particulier de patrons ODP et mis en relation avec certains types de patrons ODP. Ainsi, en rappelant qu'un *Content Pattern* décrit comment modéliser une sous-ontologie par rapport à un domaine donné, un patron de changement peut correspondre à un *Content Pattern* pour le domaine « ontologie ». De même, un *Content Pattern* peut être décrit par un patron de changement complexe composé de plusieurs changements basiques et/ou composés. L'avantage de cette mise en relation est double, le patron de

[62] http://ontologydesignpatterns.org

changement décrit comment appliquer l'ajout d'un *Content Pattern* à une ontologie existante (la composition de changement réalisant un *Content Pattern*). De plus, à travers les relations sémantiques avec les patrons d'incohérences et d'alternatives (si on considère l'ensemble complet des patrons CMP), la cohérence de l'ontologie à laquelle est appliqué le *Content Pattern* peut être maintenue. L'objectif premier d'ailleurs, de la modélisation des CMP est bien de guider le processus de gestion de changement.

Un deuxième type de relation entre les patrons CMP et les patrons ODP est défini entre les patrons d'alternatives de CMP et les patrons logiques *Logical Patterns* de ODP. Les patrons *Logical Patterns* proposent des solutions à des problèmes de conception d'ontologie que les primitives ou les constructeurs du langage de représentation ne supportent pas. Selon cette optique, les patrons d'alternatives sont proposés comme un type particulier de *Logical Patterns* permettant de résoudre des problèmes d'incohérences logiques dans la conception d'ontologies.

Le patron d'alternatives « *définir une classe hybride pour résoudre une disjonction de subsomption* » détaillé dans la section IV.3.1, et proposé comme patron *Logical Pattern* sur le portail de patrons ODP[63], a d'ailleurs été accepté dans le cadre du workshop WOP2009[64] (Workshop on Ontology Patterns), et fait l'objet d'une publication.

VII- CONCLUSION

Dans ce chapitre, nous avons présenté les patrons de gestion de changement CMP et détaillé leur spécification et leur modélisation en une ontologie OWL DL. Ainsi, la représentation formelle et explicite de la sémantique des changements, des incohérences logiques, des alternatives de résolution et de leurs relations permet de guider la gestion de l'évolution d'ontologie dans l'approche *Onto-Evo[a]l*. Les méta-connaissances fournies par les CMP, permettent d'automatiser le processus de gestion de changement et d'offrir des possibilités de recherche intelligente dans l'historique des changements notamment pour des buts d'apprentissage de nouveaux patrons.

La description de la sélection et de l'instanciation des patrons CMP au cours du processus de gestion de changement et la résolution des incohérences sont détaillées dans le chapitre suivant.

[63] http://ontologydesignpatterns.org/wiki/Submissions:Define_Hybrid_Class_Resolving_Disjointness_due_to_Subsumption

[64] http://ontologydesignpatterns.org/wiki/WOP2009:Main

Chapitre 5 : Résolution des incohérences

Résumé : Dans ce chapitre, nous détaillons la résolution des incohérences dans l'approche *Onto-Evo*^{a}*l* en expliquant comment sont sélectionnés, instanciés et employés les patrons CMP pour la gestion d'un changement, et en décrivant le procédé de résolution des incohérences et de proposition de résolutions. Une synthèse des travaux existants en débogage d'ontologie y est aussi présentée.

I- INTRODUCTION

Après avoir détaillé la modélisation des patrons de gestion de changement CMP à travers leur format de présentation ainsi que leur spécification formelle en une ontologie OWL DL, dans ce chapitre, nous expliquons comment sont appliqués les patrons CMP pour la gestion d'un changement et nous décrivons le processus de résolution des incohérences.

Le chapitre est organisé comme suit : dans la section 2, nous présentons nos besoins pour l'application des patrons CMP. La section 3 décrit l'application des patrons CMP en énonçant les principes de mise en correspondance et de sélection d'un patron CMP et en détaillant la gestion d'un changement par les patrons CMP. La section 4 est consacrée à la gestion des incohérences : nous y présentons d'abord, une synthèse des travaux existants en débogage d'ontologie, et nous détaillons le procédé de gestion d'incohérences et de proposition de résolutions guidé par les patrons CMP. Avant de conclure le chapitre, une discussion est présentée à la section 5.

II- BESOINS LIES A L'APPLICATION DES PATRONS CMP

Les principaux travaux existants en application de patrons d'ontologies se focalisent particulièrement sur l'utilisation de patrons de conception d'ontologie de domaines et appliquent des appariements terminologiques entre les éléments ontologiques à intégrer à une ontologie de domaine et les entités (concepts et relations) définis par les patrons de conception proposés pour ce domaine.

L'objectif de l'application des patrons CMP n'étant pas la simple réutilisation des patrons en tant que tels mais plutôt, le guidage du processus de gestion de changement, nos besoins sont différents de ceux liés à l'utilisation de patrons de conception d'ontologies. En fonction des phases du processus d'évolution *Onto-Evo*^{a}*l*, nous avons parfois besoin de sélectionner et instancier le patron de changement spécifiant une demande de changement et l'adapter à son contexte d'application dans l'ontologie de domaine (phase de spécification), parfois besoin de classer les incohérences détectées en sélectionnant et instanciant les patrons d'incohérences qui leur correspondent (phase d'analyse) afin de les expliciter et parfois besoin de générer des patrons d'alternatives en fonction du patron de changement et des patrons d'incohérences instanciés et de les adapter à leur contexte d'application dans l'ontologie de domaine.

La sélection des patrons CMP revient à identifier des mises en correspondance entre des parties relatives à l'ontologie de domaine (des sous-hiérarchies avec notamment des sous-hiérarchies localisées par projection de l'interprétation des résultats des raisonneurs, et des axiomes) et des parties relatives aux

patrons CMP qui correspondent également à des sous-hiérarchies et des axiomes – définissant un patron – extraits de l'ontologie CMP.

III- APPLICATION DES PATRONS CMP

L'application des patrons CMP se base sur l'identification d'appariements de classes, de propriétés et d'axiomes. Elle s'appuie sur les principes des techniques structurelles et sémantiques d'alignement de schémas d'ontologie en combinaison avec des mécanismes de raisonnement. Reprenant les principes de ces techniques d'alignement et non les techniques en elles-mêmes, nous ne développons pas dans cette section, le détail de ces travaux. Cependant, le lecteur peut se référer à l'annexe F pour une classification de ces techniques et une synthèse de leurs caractéristiques.

III.1- Principe de mise en correspondance et sélection d'un patron CMP

Les principes appliqués pour la mise en correspondance des patrons CMP sont synthétisées par le tableau ci-après (table 1):

Mise en correspondance et sélections d'un patron CMP	
Inputs	- Une sous-hiérarchie et des axiomes provenant de l'ontologie de domaine - Une sous-hiérarchie et des axiomes provenant de l'ontologie CMP - Le modèle OWL DL
Processus	- Mise en correspondance d'une sous-hiérarchie de classes de l'ontologie de domaine avec une sous-hiérarchie de classes de l'ontologie CMP décrivant un patron. Le matching ne se base pas sur la sémantique de la connaissance exprimée par une classe (ou propriété) de l'ontologie de domaine mais sur sa position, son rôle dans la hiérarchie et la nature (type) des axiomes qui lui sont appliqués, par rapport à la description du patron en cours de comparaison (section III.2).
Output	- cardinalité 1-1 (une seule possibilité d'instanciation de patron) - Equivalence ou subsomptions dans certains cas (en fonction de l'abstraction des arguments. Une superclasse par exemple, est un argument plus précis qu'un ascendant) - Degré de confiance absolu (correspondance totale)
Granularité	- Interprétation des inputs aux niveaux entité et structure tout en se basant sur la sémantique du modèle OWL DL - Granularité fine pour assurer une sélection appropriée du patron et par conséquent, la fiabilité du guidage du processus de gestion

Table 1. Principes de mise en correspondance des patrons CMP.

Les hiérarchies et les axiomes entre les entités des hiérarchies représentent des indicateurs majeurs dans la mise en correspondance des patrons CMP. Les différents aspects considérés pour le matching des sous-hiérarchies et des axiomes sont illustrés par la figure 1.

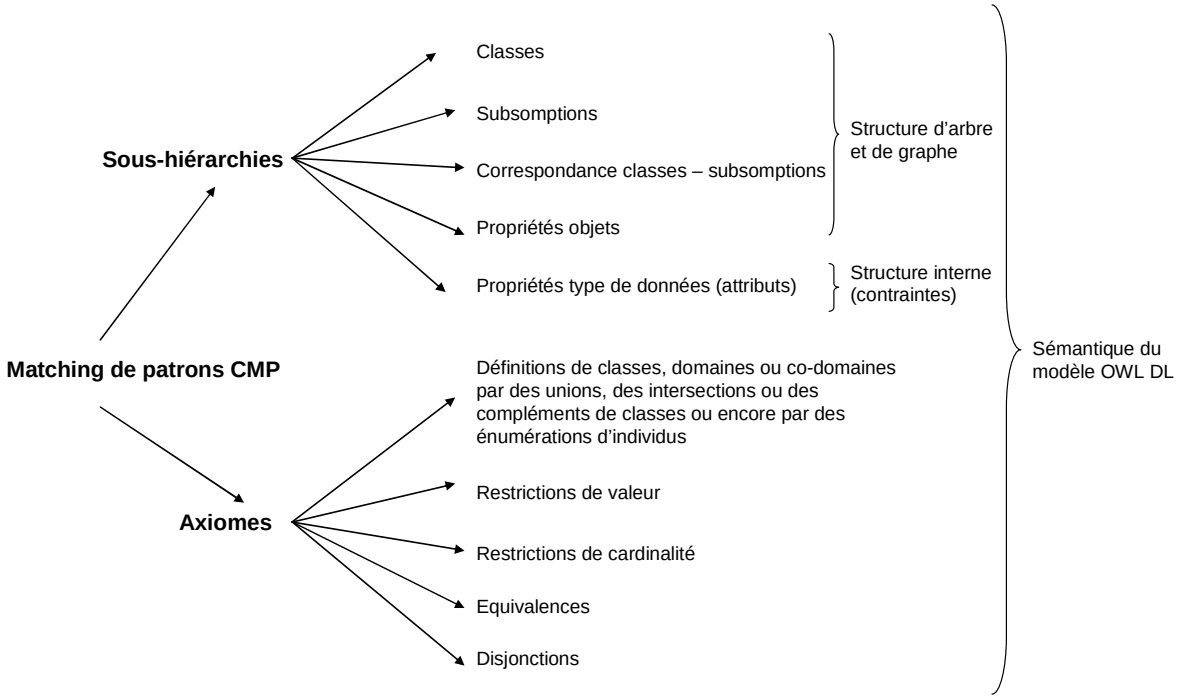


Figure 1. Matching des sous-hiérarchies et des axiomes.

III.2- Gestion d'un changement par les patrons CMP

L'application d'un patron CMP au cours du processus d'évolution, implique la sélection du patron approprié, son instanciation et son adaptation à son contexte d'application (figure 2).

Pour les patrons de changements et les patrons d'incohérences ceci inclut notamment, la mise en correspondance et la sélection du patron en important la sous-ontologie qui le définit dans l'ontologie CMP (la sous-hiérarchie dont la racine est la classe patron correspondant), et l'instanciation des entités et des axiomes définis par la description du patron.

Pour les patrons d'alternatives ceci inclut la génération du patron d'alternatives correspondant à la résolution (c'est-à-dire à l'instance du patron de changements et l'instance du patron d'incohérences), et l'instanciation des entités et des axiomes définis par la description du patron en les projetant sur les entités et les axiomes existants dans l'ontologie de domaine ou en les ajoutant. Les parties du patron d'alternatives correspondant à des entités ou axiomes déjà existants dans l'ontologie de domaine, permettent de situer le contexte d'application de l'alternative instance du patron.

Pour chaque patron CMP, une *interface* est définie pour guider la mise en correspondance et spécifier ce qui est instanciable dans le patron.

- L'interface d'un patron de changements correspond à ses arguments. Elle définit les entités manipulées par le changement, existantes dans l'ontologie de domaine, des entités créées par le changement (opérations d'ajout), des valeurs et autres paramètres en fonction du patron ;
- L'interface d'un patron d'incohérences inclut des arguments et aussi des axiomes permettant d'identifier ou de confirmer les axiomes concernés et/ou responsables de l'incohérence détectée et ainsi, de l'expliciter ;
- Outre ses arguments, l'interface d'un patron d'alternatives inclut également, les pré-conditions permettant de vérifier son applicabilité à l'ontologie de domaine.

A chaque instanciation d'un patron de changements ou d'alternatives, l'instance du patron est appliquée à l'ontologie de domaine et sa trace correspondante est enregistrée dans le journal d'évolution. Par contre, pour le patron d'incohérences seule la trace dans le journal d'évolution est enregistrée, il n'y a pas d'application de l'instance dans l'ontologie de domaine puisque l'instance du patron correspond à une incohérence détectée dans l'ontologie.

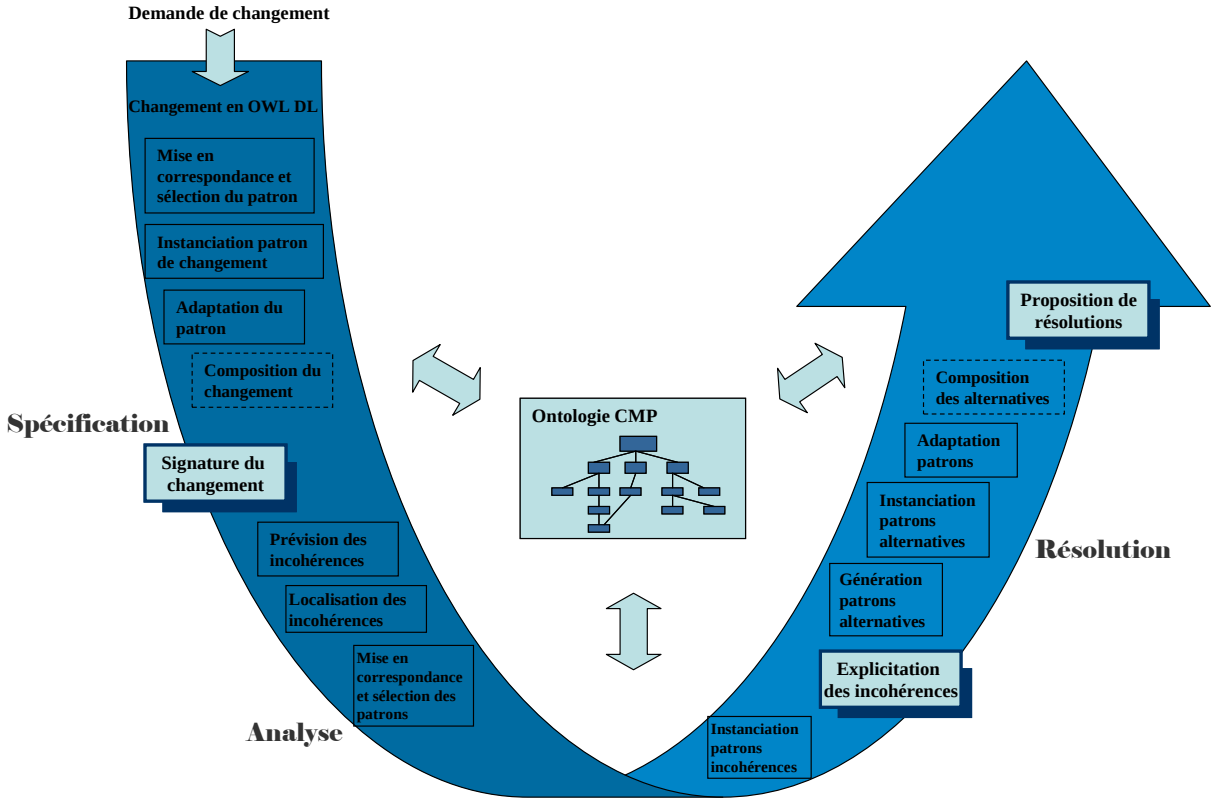


Figure 2. Gestion d'un changement par les patrons CMP.

III.2.1- Application d'un patron de changements

La mise en correspondance d'un patron de changements se prépare à l'interface de saisie d'une demande de changement avec des champs ciblant les propriétés de description des patrons de changements. L'application d'un patron de changements est illustrée par la figure ci-après (figure 3).

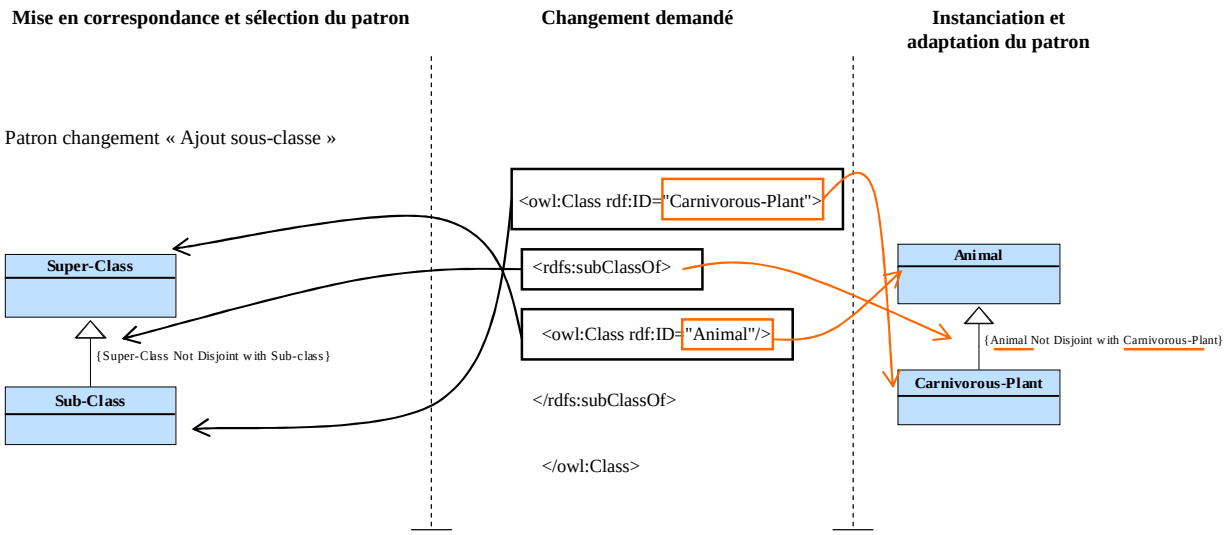


Figure 3. Application d'un patron de changements.

III.2.2- Application d'un patron d'incohérences

Le matching des incohérences détectées avec les patrons d'incohérences consiste à identifier la correspondance avec les patrons en considérant leur type et leur structure ainsi que la sémantique des éléments les constituant, représentés par l'interface des patrons d'incohérences. Guidé par le type de changement de départ et ses contraintes, nous ciblons en premier, dans la mise en correspondance, les patrons d'incohérences pouvant être causés par le type du changement traité.

Deux types d'indicateurs sont considérés (voir algorithme table 2): les informations structurelles (matching de sous-hiérarchies) et les informations axiomatiques liées aux incohérences détectées et aux descriptions des patrons d'incohérences : Pour chaque incohérence détectée, le résultat du raisonneur est d'abord enrichi par des informations d'interprétations (contexte de localisation de l'incohérence, l'entité ou les entités déclarée(s) incohérente(s), les axiomes liés à ces entités, etc.). Ensuite, l'identification de matchings de sous-hiérarchies avec les patrons d'incohérences est lancée en se basant sur les classes, les subsomptions, les attributs et les propriétés objets impliquées dans les incohérences détectées (*Matching_entité()*) et leur voisinage direct (*Correspondance_voisinage()*). Les mises en correspondance obtenues sont alors complétées par des matchings entre les informations axiomatiques des incohérences détectées et des patrons d'incohérences en se basant sur les types des axiomes qui leur sont associés (*Matching_axiome()*). La mise en correspondance des axiomes se fait en remplaçant d'abord, les classes, les propriétés et les individus déjà mis en correspondance, puis en vérifiant que l'axiome est *vrai*. La vérification des axiomes définis dans par le patron, dans l'ontologie est assurée par appel du raisonneur Pellet.

Algorithme Mise en correspondance et sélection[65]

Inputs : sous-ontologie incohérente *O*, patron d'incohérences P_{inc}

```
alignement ← ∅          // ensemble des mises en correspondance validées

// 1ère partie du matching de sous-hiérarchies : les entités
Pour chaque e_p ∈ P_inc faire    // e_p entité de type classe, propriété ou individu du patron P_inc
   appariement_entité ← ∅
   Pour chaque e_o ∈ O faire
      Si Matching_entité(e_p, e_o) = true alors      // e_o entité de l'ontologie O du même type que e_p
         appariement_entité ← appariement_entité ∪{(e_p, e_o)}
      Fin si
   Fin pour
Fin pour

// 2ème partie du matching de sous-hiérarchies : le voisinage
appariement_voisinage ← ∅
Pour chaque (e_p, e_o) ∈ appariement_entité faire
   appariement_voisinage ← appariement_voisinage ∪ Correspondance_voisinage((e_p, e_o))
   Pour chaque h ∈ appariement_voisinage faire

      // matching d'axiomes pour les sous-hiérarchies mises en correspondance
      Si Matching_axiome(h) = true alors
         alignement ← alignement ∪ {apparié P_inc}  // apparié P_inc  axiomes appariés de P_inc
      Fin si
   Fin pour
Fin pour

Si P_inc ⊑ alignement alors    // mise en correspondance complète du patron d'incohérences
  Sélectionner(P_inc)
Fin si
```

Table 2. Algorithme de mise en correspondance et sélection d'un patron d'incohérences.

Contrairement à l'instanciation d'un patron de changement qui définit des opérations et des entités à modifier ou supprimer et aussi, à ajouter dans l'ontologie de domaine pour réaliser le changement requis, et contrairement à l'instanciation d'un patron d'alternatives qui génère une résolution puis la projette sur l'ontologie de domaine, l'instanciation d'un patron d'incohérence *PInc* par une incohérence détectée *Inc* ne peut être confirmée que si une correspondance complète est retrouvée entre leurs structures et informations axiomatiques respectives :

a- A chaque classe c_{inc} dans l'incohérence *Inc* correspond une classe c_{PInc} dans la description du patron P_{Inc},

b- A chaque propriété p_{inc} dans l'incohérence *Inc* correspond une propriété p_{PInc} dans la description du patron P_{Inc}, dont le domaine correspond à celui de p_{PInc} (comme décrit dans a) et le co-domaine correspond à celui de p_{PInc} (comme décrit dans a) ou est compatible à celui de p_{PInc} si c'est une propriété type de données,

[65] Voir un exemple de trace d'exécution à l'annexe G.

c- A chaque axiome a_{inc} dans l'incohérence *Inc* (restriction de valeur, disjonction, etc.) correspond un axiome a_{PInc} dans la description du patron P_{Inc} et chaque axiome supplémentaire dans la description du patron P_{Inc} peut être déduit dans l'ontologie après remplacement des classes et des propriétés correspondantes (comme décrit dans a et b).

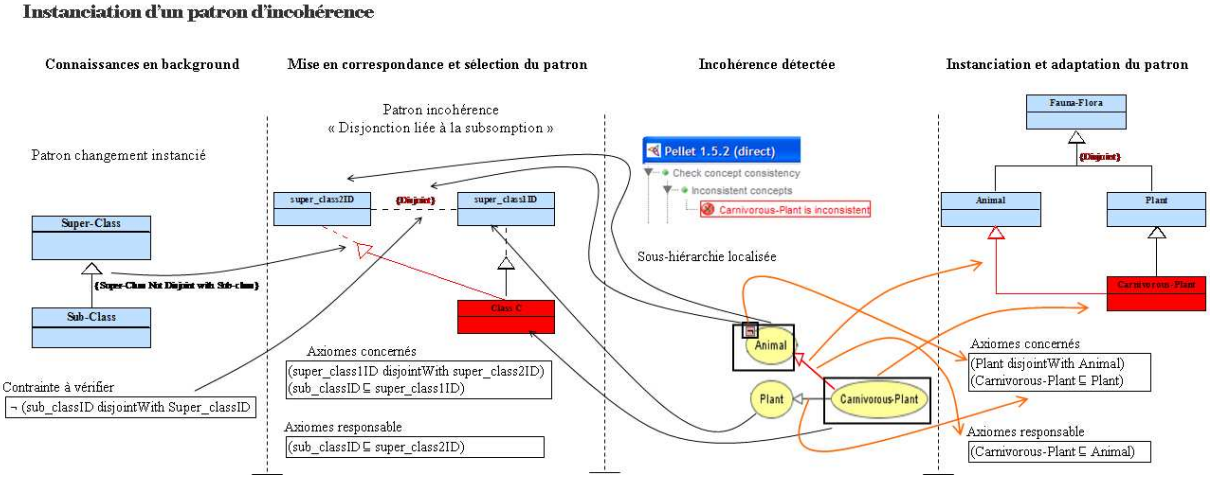


Figure 4. Application d'un patron d'incohérences.

L'algorithme de mise en correspondance doit s'assurer de la sélection du patron approprié afin de garantir – par la suite – une instanciation correcte du patron. Les contraintes d'instanciation sont vérifiées au niveau de l'interface du patron pour vérifier son applicabilité. Cette application contrôlée des patrons permet d'éviter des utilisations impropres des patrons (choix du patron invalide, correspondance et/ou instanciation partielle(s), etc.).

III.2.3- Application d'un patron d'alternatives

L'application de patrons d'alternatives comprend l'identification et la génération des patrons d'alternatives correspondant aux résolutions et l'adaptation des entités (classes, propriétés et instances) et des axiomes de chaque patron sélectionné à la partie de l'ontologie liée à l'application du changement requis et à la localisation de l'incohérence à résoudre.

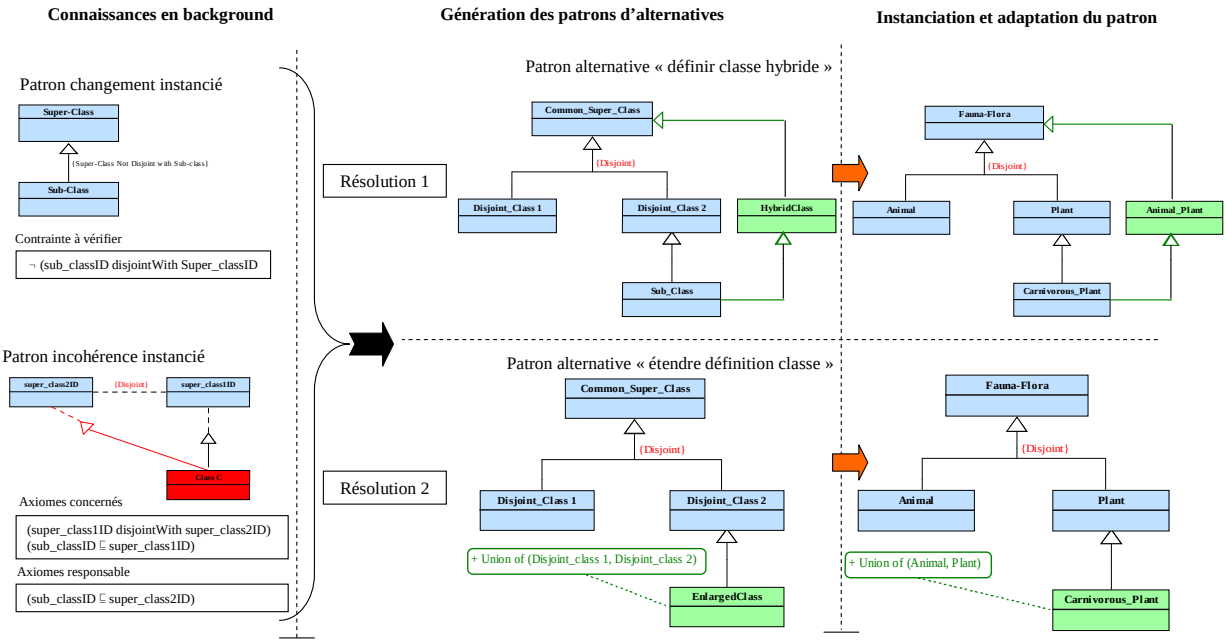


Figure 5. Application d'un patron d'alternatives.

IV- GESTION DES INCOHERENCES

Comme nous l'avons précisé dans le chapitre 3 (sections II.1.3 et VI.1), la cohérence logique gérée par l'approche *Onto-Evo*[a]*l*, se rapporte à la satisfaction des axiomes de l'ontologie au niveau logique et tient compte de la sémantique exprimée par les logiques de description et des interprétations au niveau instance.

Ainsi, une incohérence logique causée par un changement peut correspondre à :

- Une incohérence au niveau de la TBox (niveau terminologique de l'ontologie) se référant à une insatisfiabilité de concept. Elle est détectée lorsque l'interprétation d'un concept par rapport à la terminologie modélisée, est vide dans tout modèle de la terminologie.
  Exemple : Patron d'incohérences « Incohérence de disjonction liée à une subsomption» (Chapitre 4, table 17).
- Une incohérence au niveau de la TBox et la ABox (niveau terminologique et assertionnel de l'ontologie). Elle est détectée lorsque l'ontologie ne peut avoir aucun modèle de la terminologie modélisée.
  Exemple : Patron d'incohérences « Incohérence de disjonction liée à une instanciation» (Chapitre 4, figure 6). Notons que si la classe de subsomption concernée par un patron d'incohérences « Incohérence de disjonction liée à une subsomption», possède des instances dans la ABox, ce patron correspondra alors à une incohérence au niveau TBox-ABox.

Pour éviter la confusion entre toutes ces notions de cohérence due à la différence entre la terminologie anglophone et la terminologie francophone, notons qu'en anglais deux notions sont distinguées *coherence* et *consistency* pour lesquelles une seule traduction existe en français – la cohérence – bien que le sens soit tout à fait différent :

- *Coherence* : désigne pour certaines approches d'évolution d'ontologie (chapitre 2, section, III.5.4), la satisfaction de contraintes conceptuelles qui relèvent d'une bonne conception d'ontologie. Elle correspond à ce que nous désignons dans l'approche *Onto-Evo*^{a}*l*, par « cohérence conceptuelle » (Chapitre 3, section VI.1) ;
- *Consistency* ou *Logical Consistency* : désigne la satisfaction des axiomes de l'ontologie niveau logique. Elle correspond à ce que nous désignons dans l'approche *Onto-Evo*^{a}*l*, par « cohérence logique » (Chapitre 3, section VI.1).

De plus, la terminologie anglophone adoptée par les approches de gestion d'incohérence logique distingue deux notions (Haase & Qi, 2007):

- *Incoherent Ontology* (*Unsatisfiable Concept*): désigne une incohérence logique localisée au niveau de la TBox de l'ontologie (insatisfiabilité de concept);
- *Inconsistent Ontology* : désigne une incohérence logique localisée au niveau de la TBox et la ABox.

IV.1- Synthèse des travaux existants

La vérification des incohérences et des implications (entailments) constituent les principales tâches de raisonnement sur les logiques de description (Völker et al., 2008). Les approches de gestion d'incohérences sont classées selon trois types (Völker et al., 2008) :

- Des approches de raisonnement sur des ontologies incohérentes visant à répondre de manière cohérente à des requêtes lancées sur des ontologies incohérentes. Elles sont appliquées lorsque qu'il n'est pas possible de garantir la cohérence de l'ontologie ni de réparer les incohérences ;
- Des approches de débogage et de réparation d'incohérences impliquant un processus de diagnostic déterminant les causes réelles ou potentielles des incohérences, et un processus de réparation pour les corriger. Elles sont appliquées à des ontologies initialement incohérentes ;
- Des approches d'évolution cohérente d'ontologies gérant les changements d'ontologie tout en préservant la cohérence. Elles sont appliquées sur des ontologies initialement cohérentes et impliquent le contrôle des changements d'ontologie. L'approche *Onto-Evo*^{a}*l* s'inscrit dans cette optique.

Nous nous intéressons aux approches d'évolution cohérente d'ontologie et de débogage et réparation d'incohérences.

Evolution cohérente d'ontologies. Les approches d'évolution d'ontologie ayant été détaillées dans le chapitre 2, nous rappelons dans cette section, les deux principales approches d'évolution cohérente d'ontologies OWL.

- Dans (Plessers & De Troyer, 2006), les auteurs ont définit une approche et un algorithme de localisation des axiomes causant les incohérences et ont proposé un ensemble de règles que l'ingénieur d'ontologie peut appliquer pour résoudre les incohérences (Chapitre 2, section III.6).
- Dans (Haase & Stojanovic, 2005), une approche d'évolution d'ontologie OWL lite cohérente a été proposée (Chapitre 2, section III.4.3.2). Des stratégies de résolution d'incohérences ont été introduites pour résoudre des incohérences logiques.

Débogage et réparation d'incohérences. Le débogage d'ontologie est un axe de recherche très prometteur face au problème de résolution d'incohérences. La maintenance de la cohérence ne peut être assurée sans une vérification formelle permettant de délimiter les incohérences et les expliquer rationnellement. Les raisonneurs existants sont plus ou moins précis dans leurs analyses et ne donnent pas suffisamment de détails. Le débogage d'ontologie apporte un complément à leurs résultats.

L'expressivité des logiques de description offre des mécanismes de raisonnement permettant de dériver des connaissances implicites (inférer des instanciations et des subsomptions) et de détecter des erreurs telles que des concepts insatisfiables ou des implications non désirées (telles que des subsomptions inférées par le raisonneur ou définies par un processus d'apprentissage). Ces mécanismes sont appelés des *services standards de raisonnement*. Les approches de débogage apportent un complément à ces services en définissant des mécanismes capables de déterminer les justifications expliquant les erreurs détectées. Elles sont appelées des *services non standards de raisonnement*.

Les approches de débogage peuvent être classées en deux catégories (Suntisrivaraporn et al., 2008):

- Les approches *black-box* considérant le raisonneur comme une boite noire (pas de modification de comportement) et l'utilisant pour tester des requêtes sur des implications (entailments). Les incohérences sont localisées par application d'inférences principalement en employant un algorithme d'*élagage naïf* qui balaye les axiomes contenus dans l'ontologie et vérifie pour une implication si elle est conservée après suppression de chaque axiome. N'étant pas dépendantes du comportement des raisonneurs, ces approches s'implémentent facilement ;
- Les approches *glass-box* modifiant le fonctionnement interne d'un algorithme de raisonnement pour expliciter les incohérences et compléter les résultats des raisonneurs. Généralement ces modifications consistent à ajouter des labels durant le calcul pour garder la trace des axiomes correspondant aux justifications (explications des contradictions).

La comparaison de ces deux méthodes peut être synthétisée par le tableau ci-après (table 3) :

Les techniques *glass-box*	Les techniques *black-box*
- L'implémentation dépend du raisonneur et donc des techniques de raisonnement sur lesquelles il se base.	- Implémentation indépendante du raisonneur.
- Ces techniques nécessitent des modifications significatives et approfondies du comportement interne du raisonneur.	- Ces techniques ne touchent pas au comportement interne du raisonneur. Il est considéré comme une boite noire, ce qui facilite l'implémentation de ces techniques.
- Le calcul de toutes les justifications d'une implication pourrait provoquer la saturation du graphe d'achèvement (voir explication section IV.2.2) et nécessiter des optimisations pour supporter les opérations de retour arrière (roll-back) (Horridge et al., 2009).	- La vérification des implications est assurée par appel du raisonneur. Seule l'implémentation de quelques procédures répondant à des objectifs spécifiques de calcul de justification (voir explication ci-après) pourrait être nécessitée. Les procédures implémentées incluent généralement des stratégies d'optimisation (Horridge et al., 2009).

Table 3. Comparaison des techniques de débogage *black-box* et *glass-box*.

Certains travaux récents proposent des approches hybrides combinant les deux techniques *black-box* et *glass-box* (Kalyanpur et al., 2007).

IV.1.1- Approches de débogage d'ontologie

Dans (Schlobach & Cornet, 2003), une approche de débogage basée sur des services de raisonnement non standards a été proposée. Ces services ont pour objectif de localiser (to pinpoint) des contradictions dans des terminologies médicales exprimées en logiques de description. Le principe de localisation consiste à identifier des ensembles minimaux d'axiomes – appelés les justifications – dont la suppression ou l'ignorance permet de rétablir la cohérence de l'ontologie (Schlobach, 2005). Deux concepts sont définis pour déterminer les axiomes justifiant l'incohérence (Schlobach & Cornet, 2003): MUPS (Minimal Unsatisfiability Preserving Sub-TBoxes) et MIPS (Minimal Incoherence Preserving Sub-TBoxes). Cette technique a été employée dans une perspective de « clarification sémantique » d'ontologies Web enrichies automatiquement. Elle a montré ses preuves en localisation des ambiguïtés sémantiques mais ne permet pas pour autant de les résoudre.

Dans (Parsia et al., 2005), les auteurs s'intéressent au débogage d'ontologie OWL. Le débogage des classes insatisfiables et des contradictions, détaillé dans (Kalyanpur et al., 2007), se base sur la justification des implications (entailments) OWL DL. Une justification d'une implication est un sous-ensemble minimal d'axiomes de l'ontologie qui suffit pour réaliser l'implication. Elle est définie formellement comme suit (Kalyanpur et al., 2007):

Soit $O \models \alpha$ où α est un axiome, et O une ontologie cohérente. Un fragment $O' \subseteq O$ est une justification pour α dans O, dénoté JUST(α,O), si $O' \models \alpha$, et $O'' \not\models \alpha$ pour tout $O'' \subset O'$.

Ainsi, déterminer toutes les justifications d'une implication revient à identifier tous les ensembles minimaux d'axiomes responsables de l'implication. Dans (Kalyanpur et al., 2006a), les justifications des incohérences OWL ont été optimisées par un algorithme qui réduit les axiomes expliquant une incohérence à des sous-parties de ces axiomes réellement impliquées dans l'incohérence. Elles sont appelées des justifications *précises*.

L'approche est implémentée pour le raisonneur Pellet. Elle combine une implémentation *black-box* et une implémentation *glass-box* pour calculer toutes les justifications. Les justifications sont proposées comme des explications améliorant la compréhensibilité d'ontologies larges et complexes.

Une autre approche en débogage d'ontologie OWL DL est présentée dans (Wang et al., 2005). Elle se base sur une méthode heuristique permettant d'identifier les causes d'une insatisfaction de concept mais sans les résoudre. L'objectif est d'offrir à l'ingénieur d'ontologie des explications plus compréhensibles des incohérences que celles fournies par les raisonneurs standards.

Dans (Suntisrivaraporn et al., 2008), les justifications des implications sont proposées comme un service de raisonnement pour le débogage d'incohérences. De plus, les auteurs s'intéressent aux questions concernant l'efficacité et la mise en échelle de tels raisonnements pour de larges ontologies et définissent une méthode limitant l'espace de recherche des justifications à de petits modules. Le principe de *modularisation* est de regrouper les axiomes justifiant une implication dans un même module. Les auteurs démontrent que le module minimal (couvrant toute les justifications) basé sur la localité est la subsomption c'est-à-dire, qu'il suffit de se focaliser sur les axiomes contenus dans un module de subsomption pour déterminer toutes les justifications liées à une incohérence de subsomption.

Dans (Qi et al., 2006), les auteurs proposent une approche de gestion d'incohérences OWL DL basée sur la révision des croyances en complément aux travaux de (Flouris et al., 2005) (Chapitre 2, section III.5). La révision des croyances se réfère à l'adaptation cohérente d'une base de connaissances à de

nouvelles connaissances. La révision se focalise uniquement sur les incohérences dues à l'introduction de nouveaux individus dans la ABox. L'algorithme emploie deux opérateurs de révision: un opérateur de diminution/affaiblissement (weakening operator) et un deuxième opérateur correspondant à un raffinement du premier. Les opérateurs sont appliqués sur des axiomes d'inclusion de concepts (General Concept Inclusion) pour les affaiblir en définissant des exceptions explicites (le nombre d'exceptions correspond au degré d'affaiblissement d'un axiome). Le principe est que lorsqu'un axiome d'inclusion de concept cause une contradiction, plutôt que de le supprimer complètement de la ABox, l'application des opérateurs permet de diminuer de cet axiome, l'ensemble des individus causant la contradiction.

Dans (Moguillansky et al., 2008), une approche théorique de débogage d'ontologie à travers un Framework d'argumentation dynamique, basé sur les logiques de description est présentée. L'objectif de la méthodologie est de rattacher des concepts d'ontologie à des notions d'argumentation et d'employer la sémantique des argumentations pour rétablir la cohérence de l'ontologie.

IV.1.2- Outils de débogage d'ontologie

Les outils de débogage découlent des premières fonctionnalités d'explication de contradictions intégrées à des outils de construction et d'édition d'ontologie comme SWOOP (voir ci-après). Ces fonctionnalités sont restées à l'échelle de services basiques d'explication de contradictions pour certains éditeurs et navigateurs d'ontologies tels que OWLSight (voir ci-après), et ont été étendues dans des implémentations plus développées, donnant lieu à de véritables outils de débogage, munis de services de diagnostic et de réparation, et employés pour différentes applications : révision d'ontologie, fusion d'ontologies, alignement d'ontologie, etc.

OWLSight[66]

OWLSight n'est pas un outil de débogage mais un navigateur Web d'ontologie basé sur le raisonneur Pellet. La navigation dans l'arborescence des classes et des propriétés permet de visualiser également les classes et les propriétés inférées. Pour chaque entité inférée, l'outil génère des justifications et fournit des explications (explication des axiomes inférés par des axiomes affirmés) à l'utilisateur final.

Pour expliquer par exemple, l'inférence de la classe *Cheeseypizza* comme sous-classe de la classe *Pizza* dans l'ontologie *pizza*, l'outil fournit l'explication suivante :

Axiome: `CheeseyPizza subClassOf Pizza`
Explanation: `CheeseyPizza equivalentClass (Pizza and (hasTopping some CheeseTopping))`

RaDON[67]

RaDON (Repair and Diagnosis for Ontology Networks) est un plugin pour NeOn toolkit[68], un environnement extensible d'ingénierie d'ontologies en réseaux (Networked Ontologies). RaDON permet de diagnostiquer et de réparer des insatisfiabilités et des incohérences d'ontologies centralisées ou en réseaux (particulièrement des ontologies liées par des mappings). Deux modes de réparation sont possibles (Ji et al., 2009):

- Une réparation manuelle où l'utilisateur basé sur les résultats de diagnostic, choisit les axiomes à supprimer pour restaurer la satisfiabilité de concepts ou la cohérence de l'ontologie ;

[66] http://pellet.owldl.com/ontology-browser/
[67] `http://radon.ontoware.org/`
[68] http://www.neon-toolkit.org/

- Une réparation automatique en calculant toutes les explications possibles pour les insatisfiabilités de concepts diagnostiquées niveau TBox ou les incohérences diagnostiquées au niveau TBox-ABox.

SWOOP[69]

SWOOP (Semantic Web Ontology Editor) est un outil de création, édition et débogage d'ontologie OWL, initialement développé par le laboratoire Mindswap (Maryland Information and Network Dynamics Lab Semantic Web Agents Project) de l'Université de Maryland avant d'être mis à disposition sous forme d'un projet open-source. Il permet de calculer les justifications d'une insatisfiabilité de concept et de la réparer mais ne traite pas la cohérence au niveau de la ABox (Kalyanpur et al., 2005) (Kalyanpur et al., 2006a) (Kalyanpur et al., 2006b) (Kalyanpur et al., 2006c).

Protégé 4.0[70]

La version 4.0 de l'éditeur d'ontologie Protégé, offre des fonctionnalités de calcul de justifications sur une insatisfiabilité de concept, et permet de réparer manuellement une ontologie. La localisation des racines de classes insatisfiables et la génération des justifications sont réalisées de manière automatique (Horridge et al., 2008a).

IV.2- Gestion des incohérences guidée par les patrons CMP

L'objectif de l'approche *Onto-Evo*^{a}*l* est de gérer de manière automatisée l'application d'un changement et de résoudre ses impacts sur la cohérence de l'ontologie. Ceci implique l'analyse des effets du changement sur la cohérence, la localisation des incohérences causées, et la génération de changements additionnels ou substitutifs résolvant les incohérences.

IV.2.1- Caractérisation de notre approche de résolution d'incohérence

Afin de caractériser notre approche et la positionner par rapport à l'existant, nous nous sommes basés sur les caractéristiques d'analyse d'approches de résolution d'incohérence, définies dans (Haase & Qi, 2007). Dans la table 4, nous présentons ces caractéristiques et nous classons notre approche de résolution.

Caractéristiques (Haase & Qi, 2007)	**Caractéristiques de notre approche**
Application : Certaines approches sont spécifiques à applications telles que l'évolution d'ontologie, le débogage d'ontologie, la révision d'ontologie, l'alignement, etc.	La résolution d'incohérence est spécifique à une approche d'évolution d'ontologie guidée par des patrons de gestion de changements modélisant des classes de changements, d'incohérences et d'alternatives de résolution.
Granularité de la réparation : - Supprimer complètement les axiomes causant l'incohérence ; - Agir sur une partie d'un axiome et non tout l'axiome ; - Affaiblir certains axiomes en changeant la structuration des axiomes (ordre, compositions, etc.)	Certains patrons d'alternatives de résolution agissent sur une partie d'axiome (tels que changer dans un axiome de restriction de valeur le type de la restriction de ∀ à ∃). D'autres changent la structure des axiomes définissant la sous-hiérarchie incohérente (en remontant une classe par exemple). Et d'autres suppriment des axiomes (supprimer des instanciations ou des

[69] http://code.google.com/p/swoop/
[70] http://protege.stanford.edu/

	subsomptions communes à l'ajout d'une disjonction entre deux classes).
Préservation de la structure : Selon que les méthodes de diagnostic préservent ou non la structure des hiérarchies, certains algorithmes appliquent avant la réparation, des transformations de prétraitement telles que des normalisations.	Pas de transformations, la structure est préservée.
Support de connaissances terminologiques et assertionnelles : Certaines méthodes ne supportent que la TBox, d'autres la TBox et la ABox, et d'autres ne les distinguent pas.	Les deux niveaux TBox et ABox sont supportés par l'approche.
Incohérence ABox-TBox vs. Incohérence TBox : Le niveau des incohérences traitées.	Nous tenons compte aussi bien des incohérences au niveau de la TBox en réparant les insatisfiabilités de classes que les incohérences au niveau TBox-ABox lorsque les individus sont impliqués.
Complexité : Certaines approches de gestion des incohérences augment la complexité du raisonnement sur les DL exprimées par l'ontologie. Deux complexités sont à prendre en compte : - La complexité des logiques supportées - La complexité de l'algorithme de gestion des incohérences.	- Logique supportée : celle de OWL DL ($\mathcal{SHOIN}(\mathcal{D})$) - Complexité des algorithmes : -- Algorithme de localisation (incluant l'appel du raisonneur un nombre linéaire de fois) (table 5) : *PSPACE*[71] -- Algorithme de mise en correspondance des patrons d'incohérences[72] (appariement de sous-hiérarchies) (table 2) : $O(n,m)$[73] avec n et m le nombre d'entités dans les sous-hiérarchies comparées
Support de bases de connaissances multiples / en réseau	Contexte local d'évolution d'une ontologie.
Exploitation de connaissances en background ou de contexte : Considérer en input juste le contenu de la base de connaissance elle-même ou avec des informations additionnelles sur la pertinence, l'importance de certaines parties de la base de connaissances (fiabilité des sources de connaissances, argumentation de certains axiomes, etc.).	Nous ne considérons que l'ontologie elle-même.

[71] Classe des problèmes décidables par une machine déterministe en espace polynomial par rapport à la taille des données. Estimation évaluée en tenant compte des algorithmes de même caractéristiques dans (Haase & Qi, 2007).
[72] Les résolutions sont générées par instanciation des patrons d'alternatives adéquats dans l'ontologie CMP.
[73] Complexité linéaire. Estimation en comparaison avec la complexité des algorithmes d'alignement basés sur les mêmes types de techniques appliquées, et les travaux de (Herrbach et al., 2007) analysant la complexité de l'alignement de structure.

Interactivité, implication de l'utilisateur : - Procédures complètement automatiques - Intervention de l'utilisateur pour décider de certaines situations : -- Diagnostic automatique, réparation manuelle ; -- Diagnostic et proposition automatiques. Choix de la décision par l'utilisateur. -- Diagnostic et réparation automatiques. Intervention de l'utilisateur si pas de solution.	Diagnostic et réparation automatiques. Intervention de l'utilisateur si pas de solution. Si aucune alternative cohérente ne peut être proposée ou si toutes les alternatives ont un impact négatif sur la qualité de l'ontologie (chapitre 6), l'utilisateur est sollicité. Sinon, la résolution des incohérences et l'application du changement sont automatiques.
Disponibilité des implémentations : - Implémentée et disponible - Faisable, à implémenter	Implémentation partielle (prototype) (chapitre 7).

Table 4. Caractérisation de notre approche de résolution d'incohérence.

IV.2.2- Emploi d'un raisonneur DL

Un raisonneur est une implémentation d'une procédure de décision qui permet d'inférer des connaissances et de vérifier la cohérence d'une base de connaissance. Les raisonneurs DL apportent un support aux opérations de classification de base (inférer des subsomptions implicites et compléter la hiérarchie de sous-classes de l'ontologie de même qu'inférer des relations d'instanciation entre classes et individus de l'ontologie) et permettent également de vérifier la cohérence logique des ontologies. Ils implémentent généralement des algorithmes tableaux (Völker et al., 2008). La plupart des raisonneurs ne supportent pas le diagnostic et la résolution des incohérences. Ces fonctionnalités relèvent des outils de débogage.

Ils existent plusieurs raisonneurs DL parmi lesquels :

- FaCT++[74] dont les mécanismes de raisonnement sont limités à la TBox d'une ontologie OWL DL.
- Racer[75] (Renamed ABox and Concept Expression Reasoner) un raisonneur OWL/RDF et logiques modales dont la version complète est commerciale (RacerPro). Il détecte les incohérences logiques au niveau de la TBox et la ABox.
- Pellet[76] est un raisonneur OWL DL open-source. Les algorithmes tableaux qu'il implémente, permettent de vérifier la satisfiabilité de la TBox et la cohérence de la ABox par rapport à la TBox (Sirin & Parsia, 2004). La vérification d'une ABox par rapport à une TBox s'appuie sur une technique appelée *graphe d'achèvement* (Graph completion) qui tente de construire un modèle de commun de la TBox et la ABox. Un graphe d'achèvement est un graphe orienté où chaque nœud est nommé par un ensemble de concepts et chaque arrête par un ensemble de propriétés. Un prédicat binaire permet de définir des différences entre les nœuds du graphe (Horrocks & Sattler, 2007).

  Pellet fournit les justifications des inférences qu'il génère et extrait pour certaines incohérences détectées, un ensemble unique composé des axiomes ayant causé l'incohérence (Sirin et al., 2007). Cependant, il ne propose pas de résolutions.

[74] http://owl.man.ac.uk/factplusplus/
[75] http://www.racer-systems.com/
[76] http://clarkparsia.com/pellet/

Dans notre approche, nous avons employé le raisonneur Pellet pour l'analyse et la détection des incohérences. Pellet est plus ou moins précis dans ses analyses en fonction des types d'incohérences et ne donne pas toujours suffisamment de détails. En effet, certaines incohérences logiques, notamment celles se référant aux propriétés, ne sont pas détectées par Pellet et sont prises en charge par les patrons CMP. Par ailleurs, Pellet ne permet pas – pour certaines incohérences – de préciser les axiomes qui ont causé les incohérences ni comment résoudre les incohérences détectées. Il y a besoin de compléter ses résultats.

IV.2.3- Analyse de l'impact du changement

Rappelons l'objectif de la phase d'analyse du changement de l'approche *Onto-Evo[a]l* (chapitre 3, section IV.2) qui consiste à appliquer – temporairement – le changement requis dont la signature est définie par l'instance d'un patron de changements afin de vérifier la cohérence logique et de délimiter la partie incohérente de l'ontologie pour la résoudre. Cette phase se compose de deux principaux volets : la localisation des incohérences et leur classification selon les patrons d'incohérences prédéfinis par l'ontologie CMP afin de les expliciter.

IV.2.3.1- Démarche générale

La démarche générale peut être résumée comme suit (figure 2, analyse):

1- Prévision des incohérences attendues ({*P_Inc_déduits*}): Cette étape permet de vérifier les contraintes spécifiées par la description du patron de changements instancié, et de déduire à partir des contraintes du changement non vérifiées (valeur = *false*), les incohérences potentielles pouvant être causées. Ces incohérences sont définies par les patrons d'incohérences liés au patron de changements – instancié – par la propriété « *cause_potentiellement* », dans l'ontologie CMP.

2- Appel du raisonneur pour vérifier l'impact sur la cohérence. Si l'ontologie évoluée est cohérente aller à 5.

3- Localisation des incohérences ({*Inc_localisées*}): cette étape vise à détecter et localiser les incohérences réellement causées par l'application du changement (section IV.2.3.2, table 5). La spécification du changement à elle seule, ne suffit pas pour déterminer l'impact sur la cohérence logique de l'ontologie, il faut vérifier tous les axiomes se référant à l'entité ajoutée/modifiée/supprimée.

4- Mise en correspondance et sélection des patrons d'incohérences ({*P_Inc_sélectionnés*}): Cette étape permet de mettre en correspondance les patrons d'incohérences avec les incohérences localisées ({*Inc_localisées*}) afin de les classer et sélectionner les patrons correspondants (algorithme table 2).

5- Si {*P_Inc_sélectionnés*} $\subseteq$ {*P_Inc_déduits*} alors
Instancier {*P_Inc_sélectionnés*} $\cup$ {*P_Inc_déduits*}
Sinon
Instancier {*P_Inc_sélectionnés*}
Fin si.

Cette dernière étape permet de traiter le cas de certaines incohérences supportées par l'approche *Onto-Evo[a]l* et la modélisation des patrons CMP mais non détectées par les raisonneurs telles que l'*incohérence d'intersection de domaines disjoints* (chapitre 4, section VI.3, 3ème observation).

IV.2.3.2- Localisation des incohérences

La localisation des incohérences est conduite selon une approche *black-box*. L'algorithme de localisation est conçu comme une couche supérieure indépendante du raisonneur faisant appel à lui un nombre linéaire de fois. Ce choix s'accorde avec les besoins liés à l'application des patrons CMP.

L'algorithme de localisation des incohérences étend l'algorithme de localisation présenté dans (Haase & Stojanovic, 2005), permettant de déterminer la sous-ontologie *O'* incohérente minimale telle que $O' \subseteq O$ (*O* étant l'ontologie analysée) et $\forall O'' \subset O'$, *O''* est une sous-ontologie cohérente de *O*.

Le principe est de prendre les axiomes définissant le changement à appliquer – correspondant à l'instance d'un patron de changement – comme input d'une fonction de sélection appelée de manière itérative pour sélectionner un sous-ensemble plus large d'axiomes de l'ontologie afin de constituer la sous-ontologie incohérente minimale. La sélection des axiomes se base sur le voisinage structurel des axiomes considérant deux axiomes comme étant structurellement connectés, s'ils se réfèrent à des entités communes de l'ontologie. Il est défini plus formellement par la connexité structurelle (structural connectedness) énonçant que (Haase & Stojanovic, 2005): Deux axiomes α et $\beta \in$ à une ontologie *O* sont structurellement connectés (directement) – noté Connecté(α, β)– s'il existe une entité $e \in$ à *O* (e étant une classe, une propriété ou un individu) telle que *e* est contenue dans l'axiome α et l'axiome β.

Algorithme Localisation des incohérences

Input : $\{\alpha_{Ch}\}$ //axiomes correspondant au changement à appliquer

// l'incohérence peut être localisée dans le changement lui même
// si le changement altère par exemple, la cohérence de la définition de l'entité objet du changement
Si Détecter_incohérences($\{\alpha_{Ch}\}$) = *false* **alors** // cette fonction appelle le raisonneur
 $\Omega \leftarrow \{\{\alpha_{Ch}\}\}$ // ensemble des sous-ontologies candidates

Répéter
 $\Omega' \leftarrow \emptyset$
 Pour chaque $O' \in \Omega$ **faire** // *O'* une sous-ontologie candidate
  Pour chaque $\beta_1 \in O \setminus O'$ **faire**
  // β_1 axiome ∈ à *O* privée des axiomes contenu dans les sous-ontologies candidates
   Si il existe $\beta_2 \in O'$ tel que Connecté(β_1, β_2) **alors**
    $\Omega' \leftarrow \Omega' \cup \{ O' \cup \{\beta_1\}\}$
   Fin si
  Fin pour
 Fin pour

 $\Omega \leftarrow \Omega'$
 // recherche d'une ontologie candidate incohérente
 nb_candidate ← |Ω| // nombre de sous-ontologies candidates dans Ω
 cohérence ← *true*
 i ← 1
 Tant que (cohérence = *true* AND i ≤ nb_candidate) **faire**
  O' ← Ω[i]
  Si Détecter_incohérences(*O'*) **alors** //Appel du raisonneur
   cohérence ← *false*
  Sinon
   i ← i+ 1
  Fin si

Fin Tant que

Jusqu'à cohérence = *false*
Fin si
Output : Ω[i]

Table 5. Algorithme de localisation des incohérences.

Un exemple d'exécution de l'algorithme de localisation, basé sur l'ontologie *O* illustrée dans le chapitre 4, est détaillé dans l'annexe G.

IV.2.4- Proposition de résolutions

La phase de résolution du changement de l'approche *Onto-Evo*^{a}*l* (chapitre 3, section IV.3) comprend deux principales activités : la proposition de résolutions et l'évaluation des résolutions. La première activité permet de générer les patrons d'alternatives appropriés en se basant sur les résultats de la phase d'analyse, et de les instancier. La deuxième activité permettant d'évaluer l'impact des résolutions proposées sur la qualité d'ontologie, sera détaillée dans le chapitre 6.

Une fois les incohérences identifiées et classées par instanciation des patrons d'incohérences correspondants, les patrons des alternatives de résolution potentielles sont générés et instanciés (figure 5) en se basant sur la classe Résolution – de l'ontologie CMP – définissant les relations sémantiques de résolution entre le patron de changement et les patrons d'incohérences instanciés (chapitre 4, figure 9). Chaque résolution correspond à un ensemble de changements à appliquer. Elles peuvent alors causer des incohérences. C'est pourquoi toutes les résolutions sont vérifiées – en faisant appel au raisonneur – et seules les résolutions cohérentes sont retenues. La procédure de proposition de résolutions est illustrée par la figure 6.

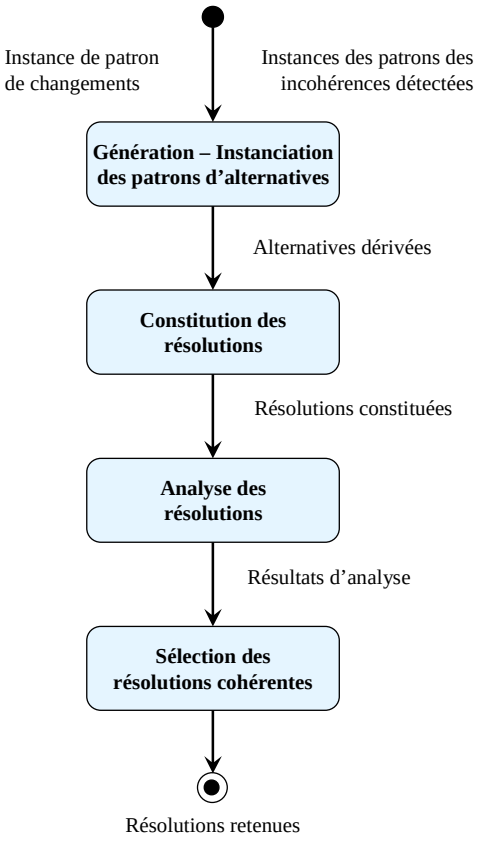


Figure 6. Diagramme d'activités de la procédure de proposition de résolutions.

Afin de résumer toutes les étapes, la gestion des incohérences peut être illustrée par le scénario suivant (table 6) :

Spécification	**Analyse**	**Proposition de résolution**			
		Alternatives dérivées	Résolutions constituées	Résultats d'analyse	Résolutions retenues
Ch_1	Inc_1	Al_1, Al_2	Res_1 (Al_1, Al_3 , Al_4)	Res_1: Cohérence	Res_1
	Inc_2	Al_3	Res_2 (Al_1, Al_3 , Al_5)	Res_2: Incohérence(s)	
	Inc_3	Al_4, Al_5	Res_3 (Al_2, Al_3 , Al_4)	Res_3: Incohérence(s)	
			Res_4 (Al_2, Al_3 , Al_5)	Res_4: Cohérence	Res_4

Table 6. Illustration d'un scénario de gestion d'incohérence.

V- DISCUSSION

Outre, la modélisation même des patrons CMP, la deuxième contribution de l'approche *Onto-Evo*^{a}*l* est l'application de ces patrons pour le guidage du processus de gestion de changement. Dans cette section nous discutons de la localisation des incohérences au niveau de la TBox et la Abox en appliquant une technique *black-box*, de l'apport des patrons d'incohérences et d'alternatives par rapport aux stratégies de débogage existantes, de l'optimisation des explications d'incohérences et de la traçabilité des instanciations de patrons.

V.1- Localisation des incohérences au niveau de la TBox et la ABox

L'approche *Onto-Evo*^{a}*l* supporte la localisation des incohérences logiques à deux niveaux : la TBox (insatisfiabilité de concept), et la TBox et ABox (aucun modèle pour l'ontologie), ce qui offre une gestion plus complète des incohérences. Une incohérence TBox-ABox peut être en effet, une conséquence potentielle d'une incohérence TBox (Flouris et al., 2006b). L'instanciation d'un concept insatisfiable engendre une incohérence au niveau de la ABox. Si nous reprenons encore une fois, l'exemple de l'ontologie *O* illustrée dans le chapitre 4 et définie comme suit :

$\{Animal \sqsubseteq Fauna\text{-}Flora,\ Plant \sqsubseteq Fauna\text{-}Flora,\ Carnivorous\text{-}Plant \sqsubseteq Plant,\ Plant \sqsubseteq \neg Animal\}$

L'ajout du changement *Ch1* correspondant à l'axiome :

$Carnivorous\text{-}Plant \sqsubseteq Animal$

Cause une incohérence au niveau de la TBox liée à l'insatisfiabilité du concept *Carnivorous-Plant*. Si le concept insatisfiable *Carnivorous-Plant* a une instance dans la ABox :

```
Carnivorous-Plant (Nepenthes Truncata)
```

L'incohérence causée est alors localisée au niveau de la TBox et la ABox.

V.2- Localisation black-box

Dans notre approche, la localisation des incohérences est indépendante du raisonneur (black-box) et fait appel à lui un nombre linéaire de fois mais, bien qu'elle soit optimisée par le commencement par les axiomes représentant le changement appliqué et par une navigation centrée sur le voisinage des axiomes, elle nécessite plusieurs appels du raisonneur ce qui augmente la complexité des calculs. En perspective, nous pensons optimiser encore plus l'algorithme pour améliorer ses performances et notamment l'adapter à la localisation d'incohérences dues à des changements complexes. Plusieurs techniques récentes d'optimisation d'algorithmes de débogage *black-box* peuvent être explorées à cet effet (Kalyanpur et al., 2007) (Suntisrivaraporn et al., 2008) (Baader & Suntisrivaraporn, 2008) (Ji et al., 2008).

V.3- Stratégies de débogage d'ontologie vs. Patrons d'incohérences et d'alternatives

Plusieurs outils et stratégies de débogage sont proposés dans la littérature (section IV.1). Ils apportent un support considérable aux développeurs d'ontologie en justifiant des inférences et expliquant des insatisfiabilités de concepts ou des incohérences. Cependant, ils offrent peu de fonctionnalités de résolution. Les solutions proposées sont toujours limitées : supprimer les axiomes causant les incohérences, ou retirer une partie de ces axiomes.

Nous soutenons qu'il serait possible de fournir un support supplémentaire particulièrement pour l'évolution cohérente d'ontologies, à travers l'identification et la modélisation de patrons communs d'incohérences causées par certains types de changements, et la proposition d'alternatives qui pourraient potentiellement les résoudre. L'application des patrons CMP en combinaison avec les fonctionnalités offertes par le raisonneur Pellet, rendent le guidage du processus de gestion de changements plus efficace.

De plus, les alternatives de résolution que nous proposons ne consistent pas à supprimer tout simplement un des axiomes constituant la sous-ontologie incohérente minimale. Elles reposent plutôt sur des solutions de généralisation de classes, d'extension de définition, etc.

V.4- Optimisation des explications d'incohérences

Les approches de débogage fournissent des explications justifiant des implications généralement inférées par des axiomes affirmés (asserted axioms) et optimisent le processus d'explication en appliquant des algorithmes qui simplifient les axiomes fournis en explication (les justifications) en prenant des sous-parties de ces axiomes (par exemple, pour une classe complexe, ils prennent juste la partie de l'axiome qui explique l'inférence basée sur cette classe et non toute sa définition). Plusieurs approches sont proposées : des justifications de faible granularité, précises ou encore laconiques (Kalyanpur et al., 2006a) (Horridge et al., 2008a).

Dans l'approche *Onto-Evo*^{a}*l* d'évolution cohérente d'ontologie, notre optique est de localiser et expliquer les impacts d'un changement (un ensemble d'axiomes ajoutés par l'utilisateur) sur la cohérence et les résoudre. L'objectif premier à la base de la modélisation des patrons CMP (Change Management Patterns) est de fournir une base de connaissances permettant de lier un ensemble de types d'incohérences à un type de changement, de faire ressortir des descriptions génériques afin de guider la localisation et l'explication des incohérences et par le même principe, proposer pour chaque type d'incohérences des alternatives types permettant de les résoudre. La généricité et l'abstraction caractérisant les patrons ne permettent pas de réfléchir à ce stade – de modélisation – à des optimisations des explications. Toutefois, l'intégration en perspectives, de méthodes de calculs optimisés de justification serait très prometteuse pour l'enrichissement des patrons d'incohérences, l'optimisation de l'algorithme de localisation et par là, l'amélioration des résolutions.

V.5- Traçabilité des instanciations de patrons CMP

La relation entre un patron CMP et son instanciation est décrite formellement comme une instance de l'ontologie CMP enregistrée dans le log d'évolution comme instance de la classe *Trace_changement*, *Trace_incohérence* ou *Trace_alternative* (Chapitre 3, section V.2). Les résolutions globales proposées pour un changement sont enregistrées dans le log d'évolution comme instance de la classe *Trace_résolution_chgt.* Même les incohérences non résolues et les résolutions de changement incohérentes sont sauvegardées afin d'assurer une traçabilité détaillée de tous les traitements et les résultats du processus, et de constituer une riche source de connaissances pour l'apprentissage de patrons.

VI- CONCLUSION

Dans ce chapitre, nous avons détaillé l'application des patrons CMP en précisant les spécificités de nos besoins et les principes de mise en correspondance de parties d'ontologie auxquels nous avons eu recours, et nous avons exposé le procédé de gestion des incohérences et de proposition de résolutions.

Dans le chapitre suivant, nous abordons l'évaluation de l'impact des résolutions sur la qualité de l'ontologie.

Chapitre 6 : Evaluation des résolutions

Résumé : Dans ce chapitre, nous détaillons l'ensemble des caractéristiques, critères et métriques définissant le modèle de qualité sur lequel se base l'évaluation des résolutions dans l'approche *Onto-Evo*^{a}*l*, et nous décrivons l'emploi de l'évaluation de qualité pour le guidage de la résolution d'un changement. Une synthèse des travaux existants en évaluation d'ontologie y est aussi présentée.

I- INTRODUCTION

L'évaluation d'ontologie est une partie du cycle de vie des ontologies. Les principales approches d'ingénierie ontologique (Fernandez-Lopez et al. 1997) (Fernandez-Lopez et al. 1999) (Sure & Studer, 2002) (Tempich et al. 2005) intègrent des activités d'évaluation dans leur processus. L'évaluation de qualité peut servir de guide lors du processus de construction et aussi dans toute étape d'apprentissage ou de raffinement d'ontologie. Elle est aussi primordiale pour l'utilisation des ontologies aussi bien dans le contexte du web sémantique que dans des applications basées sur la sémantique. Souvent, les utilisateurs doivent choisir, parmi une multitude d'ontologies, l'ontologie la plus appropriée en fonction de leurs besoins. Ce choix se base essentiellement sur le résultat de l'évaluation de chacune des ontologies.

Nous soutenons que la gestion automatisée des changements pourrait être potentiellement optimisée par l'évaluation de l'impact des résolutions proposées, sur la qualité de l'ontologie et que l'application de techniques d'évaluation de qualité – basées sur des métriques quantifiables – pourrait être d'un apport crucial pour le processus d'évolution d'ontologie.

Dans ce chapitre, nous proposons une approche complémentaire aux travaux existant en évolution d'ontologies intégrant – outre la modélisation des patrons CMP – l'évaluation de l'impact du changement sur la qualité de l'ontologie. Ainsi, plutôt que de solliciter l'intervention de l'expert dans la gestion des incohérences, nous guidons le choix des résolutions proposées en évaluant leur impact sur la qualité de l'ontologie. Le guidage du processus de gestion de changement se trouve par conséquent, optimisé.

L'évaluation de l'impact des résolutions sur la qualité se base sur des métriques d'évaluation existantes. Ainsi, nous commençons ce chapitre par une synthèse des principaux travaux existants en évaluation de qualité. Dans la section 3, nous détaillons le modèle de qualité que nous avons défini et les métriques et critères considérés. L'application de l'évaluation de qualité pour le guidage de la résolution de changement est décrite dans la section 4. Avant de conclure le chapitre, une discussion est présentée à la section 5.

II- SYNTHESE DES TRAVAUX EXISTANTS

Plusieurs travaux en évaluation d'ontologie sont proposés dans la littérature. Dans cette section, nous en présentons une synthèse sans prétendre à une étude complète de l'état de l'art, notre objectif étant d'introduire les fondements des approches existantes et les métriques réutilisées pour la définition du modèle de qualité, sur lequel se base l'évaluation des résolutions. Pour une étude plus complète, le lecteur peut se référer à (Gomez-Perez, 2004) (Hartmann et al., 2005) (Brank et al., 2005) (Supekar, 2005) (Volker et al., 2005) (Yang et al. 2006) (Vrandecic & Surez-Figueroa, 2006).

II.1- Niveaux d'évaluation d'ontologie

Par sa complexité, il est plus pratique d'évaluer l'ontologie par niveau que de la considérer dans sa globalité. Différents niveaux d'évaluation d'ontologie ont été considérés dans la littérature : le niveau lexical (vocabulaire), le niveau structurel (taxonomie), le niveau usage (l'application utilisant l'ontologie) et le niveau domaine (les données).

Le niveau lexical correspond aux concepts, propriétés et instances décrits dans l'ontologie et le vocabulaire utilisé pour les représenter. L'évaluation à ce niveau, se base sur des comparaisons par rapport aux sources représentatives du domaine. Dans (Maedche et al., 2002), les auteurs ont proposé une approche pour évaluer le vocabulaire d'une ontologie en calculant la similitude entre deux termes avec la distance de *Levenshtein* (similarité lexicale). Le contenu lexical d'une ontologie peut être évalué en utilisant la précision et le rappel souvent utilisés dans le traitement automatique des langues. Dans ce contexte, la précision représente le pourcentage des termes lexicaux de l'ontologie (chaînes utilisées pour nommer les concepts) qui apparaissent dans le standard par rapport au nombre total des mots de l'ontologie. Le rappel représente le pourcentage des termes lexicaux du standard qui apparaissant également comme noms de concepts dans l'ontologie par rapport au nombre total de termes lexicaux du standard.

La taxonomie est évaluée par différentes mesures structurelles impliquant en autres (Gangemi et al., 2005): la profondeur, la largeur, la distribution des feuilles, la densité, la modularité, la cohérence, la complexité, etc. Dans (Maedche et al., 2002), plusieurs mesures sont proposées pour comparer les aspects relationnels entre deux structures d'ontologie. Dans (Brewster *et al*, 2004), les auteurs évaluent le degré d'adéquation de la structure de l'ontologie à un corpus représentatif du domaine.

L'ontologie peut aussi être évaluée sur la base des résultats de l'application qui l'utilise. Cependant, une telle approche présente quelques inconvénients (Porzel & Malaka, 2004) : (*i*) le résultat se base sur une tâche particulière et il est difficile de généraliser les observations d'évaluations ; (*ii*) l'ontologie constitue un petit composant de l'application et son impact sur le résultat peut être faible et indirect ; (*iii*) la comparaison de plusieurs ontologies ne peut se faire que si elles sont utilisées par la même application.

L'évaluation niveau domaine consiste à comparer l'ontologie à une collection de documents ou un standard représentant le domaine d'intérêt. Dans (Patel et al., 2004), les auteurs ont montré comment déterminer si une ontologie se rapporte à un domaine d'intérêt particulier, et classifient les ontologies dans un annuaire de domaine d'intérêt. Des données terminologiques telles que les noms de concepts, sont extraites à partir de l'ontologie et utilisées comme entrée d'un modèle de classification. Ainsi, plutôt que de comparer plusieurs ontologies entre elles afin de choisir la plus appropriée à un usage particulier, cette approche propose de comparer les ontologies à un corpus. Après avoir procédé à l'extraction automatique de termes à partir du corpus, le chevauchement entre les termes obtenus par extraction et les concepts de l'ontologie sera calculé. Une ontologie peut être pénalisée par des termes disponibles au niveau du corpus et absents dans l'ontologie, comme elle peut, à l'inverse, être pénalisée par des concepts disponibles au niveau de l'ontologie et absents dans le corpus. Dans (Alani *et al*, 2006), les auteurs présentent un outil *AKTiveRank* qui permet de classer des ontologies en comparant leur pertinence par rapport à un ensemble de termes spécifiés par l'utilisateur. La technique de classement (ranking) évalue les schémas d'ontologie en se basant sur des techniques structurelles.

Certains travaux d'évaluation se basent sur une approche multi-niveaux/multicritères souvent utilisée pour la sélection de l'ontologie la plus appropriée parmi un ensemble d'ontologies ce qui se ramène à un problème de prise de décision. Cette approche est basée sur la définition d'un ensemble de critères (attributs). Pour chaque critère, l'ontologie est évaluée et un score est attribué. De plus, un poids est assigné à chaque critère. Dans (Burton-Jones *et al.,* 2004), une approche à dix critères est proposée : l'éligibilité (fréquence des erreurs syntaxiques), la richesse, l'interprétabilité (la présence des termes

utilisés dans WordNet), la cohérence (nombre de concepts impliqués dans des contradictions), la clarté (le sens des termes utilisés dans l'ontologie par rapport à WordNet), la compréhensivité, l'exactitude (pourcentage de fausses relations), la pertinence, l'autorité (combien d'autres ontologies utilisent les concepts de l'ontologie à évaluer?) et l'historique (nombre d'accès à l'ontologie). Dans (Fox *et al*, 1998), les critères considérés sont : la complétude fonctionnelle (l'ontologie contient-elle assez d'information?), la généralité (l'ontologie est-elle assez générale pour qu'elle soit partagée par plusieurs utilisateurs?), l'efficacité du raisonnement supporté par l'ontologie, la compréhension, la précision/granularité (l'ontologie supporte-t-elle plusieurs niveaux d'abstraction/de détail?) et la minimalité (l'ontologie contient-elle tous les concepts nécessaires?). Dans (Lozano-Tello *et al*, 2004), les auteurs présentent *OntoMetric*, un Framework hiérarchique d'évaluation de la qualité et de la pertinence des ontologies par rapport aux besoins utilisateurs. *OntoMetric* se base sur 117 critères couvrant cinq aspects de l'ingénierie d'ontologie : le contenu de l'ontologie (concepts, relations, taxonomie et axiomes), la méthodologie utilisée lors de la construction de l'ontologie, le coût d'utilisation de l'ontologie et les outils disponibles.

II.2- Approches d'évaluation d'ontologie

OntoClean[77] (Guarino & Welty, 2002) (Guarino & Welty, 2004) est une méthodologie d'analyse de relations taxonomiques basée sur des notions ontologiques génériques inspirées de la philosophie. Elle définit quatre principales méta-propriétés: la rigidité, l'identité, l'unité, et la dépendance, auxquelles s'ajoutent d'autres méta-propriétés définies pour catégoriser certains types de rigidité (Welty & Anderson, 2005). Ces méta-propriétés permettent de structurer les ontologies en contraignant l'utilisation des relations de subsomption, et de guider l'évaluation et la validation des choix de conceptualisation. Elles sont évaluées en se basant sur un ensemble de règles à vérifier. Les violations de règles détectées sont corrigées en remontant/descendant des classes dans la taxonomie ou en ajoutant/supprimant des classes.

Dans (Gangemi et al., 2006a), une approche multicouches est présentée. Les auteurs définissent :

- O^2 : une méta-ontologie permettant d'évaluer les ontologies comme un objet sémiotique[78]. L'ontologie est considérée selon sa dimension syntaxique (structure du graphe), sa dimension sémantique (sa conceptualisation), et sa dimension pragmatique (les configurations de communication liées aux inférences, aux annotations et à l'usage).
- *oQual* (ontology Quality), un patron basé sur O^2 modélisant l'évaluation d'ontologie comme une tâche de diagnostic ;
- *QOOD* (Quality-Oriented Ontology Description), un composant de *oQual* décrivant les critères d'évaluation considérés (Gangemi et al., 2006b). L'évaluation d'ontologie tient compte de la dimension structurelle (largeur, profondeur, chevauchement, densité, modularité, etc.) et fonctionnelle (précision, rappel, exactitude, adéquation et d'autres critères qualitatifs) de l'ontologie de même que du contexte d'usage (exhaustivité, fiabilité, etc.).

Dans (Tartir et al., 2005), les auteurs présentent un Framework *OntoQA*, d'analyse et de synthèse de contenu d'ontologie. Il a été enrichi pour permettre de classer des ontologies en fonction de mots clés et de caractéristiques ontologiques définis par l'utilisateur (Tartir & Arpinar, 2007). L'approche se base sur un ensemble de métriques de qualité qui tiennent compte non seulement du schéma de l'ontologie mais aussi des instances :

[77] http://semanticweb.org/wiki/OntoClean
[78] Par référence à l'étude de processus de signification et de communication de signes.

- Les *métriques schéma* évaluent la conception du schéma de l'ontologie (héritage, les attributs et les relations sémantiques) ;
- Les *métriques base de connaissances* évaluent les classifications des **instances** dans l'ontologie et l'utilisation effective des connaissances modélisées dans le schéma (richesse de la base de connaissances).

Le modèle de qualité que nous avons défini, réutilise un ensemble de ces métriques. Nous présenterons leur définition dans la section 3.

Dans (Vrandecic & Sure, 2007), les auteurs proposent d'enrichir l'évaluation structurelle par des métriques tenant compte de la sémantique du modèle dans lequel est exprimée l'ontologie particulièrement, OWL DL. L'approche recommande des étapes de normalisation d'ontologie en prétraitement avant l'application de métriques structurelles standards (Vrandecic & Sure, 2007): nommage des classes et individus anonymes, contrôler l'inférence des subsomptions, unification des noms, propagation des instances à la classe la plus profonde dans la hiérarchie, normalisation des instances de propriétés (inférer les propriétés inverses et symétriques, etc.). Ces étapes permettent de révéler par les transformations de normalisation, la sémantique de la structure de l'ontologie, et de faciliter par là, l'évaluation de la qualité.

Relativement aux autres approches existantes, indépendantes du modèle de l'ontologie, cette approche permet de mieux gérer les transformations de structure nécessaires parfois, au calcul de métriques structurelles mais, elles ne sont pas applicables pour toutes les métriques.

II.3- Evaluation en évolution d'ontologie

L'évaluation de qualité a son importance en évolution d'ontologie. Elle a été employée pour l'identification de besoins de changement dans des approches d'apprentissage d'ontologie (chapitre 2, section III.1) : par application d'un ensemble de métriques d'évaluation de qualité, plusieurs changements sont proposés pour améliorer et affiner l'ontologie.

L'évaluation de qualité a aussi été employée pour contrôler le processus d'évolution :

- Dans (Haase & Stojanovic 2005), quelques critères de qualité – non-quantifiables – ont été évoqués, sous forme de conditions génériques de cohérence utilisateur définies par l'ingénieur pour assurer une bonne modélisation de l'ontologie telles que les méta-propriétés de la méthodologie *OntoClean* (section II.2).
- Dans (Dividino & Sonntag, 2008), une approche d'évolution contrôlée d'ontologie basée sur des méthodes d'évaluation sémiotique, est présentée. L'outil d'évaluation *S-OntoEval* implémenté, combine un ensemble de métriques existantes de qualité, catégorisées selon trois niveaux sémiotiques pour contrôler le processus d'évolution :
    - L'évaluation sémiotique structurelle (dimension syntaxique) contrôle la représentation des changements dans le modèle de l'ontologie, et la sémantique des changements appliqués notamment par rapport à la cohérence logique de l'ontologie. Les mesures appliquées concernent la typologie et la cohérence logique par application de raisonneur ;
    - L'évaluation sémiotique fonctionnelle (dimension sémantique) contrôle les effets des changements sur la cohérence de l'ontologie par rapport à son objectif d'usage, de même que la cohérence de la propagation des changements aux artefacts dépendant de l'ontologie évoluée. Elle consiste à confronter la structure du graphe de l'ontologie et la sémantique formelle exprimée par le modèle de l'ontologie, à la sémantique cognitive attendue de sa conceptualisation, en évaluant l'adéquation de l'ontologie à une tâche spécifique comme

défini dans (Porzel & Malaka, 2004) (section II.1). L'évaluation se base sur l'observation de la performance de la tâche basée sur l'ontologie, en comparant les réponses données par l'ontologie à celles d'un standard adapté à la tâche ;

- L'évaluation sémiotique d'utilisabilité (dimension pragmatique) contrôle la validation finale du changement en évaluant le *profil* de l'ontologie sur la base des annotations qui la décrivent. Elle applique des métriques d'analyse d'annotations qui calculent le nombre de métadonnées décrivant l'ontologie pour vérifier que les changements ont été rapportés aux annotations (par augmentation du nombre d'annotations et non leur contenu).

L'évaluation de qualité a été appliquée également pour évaluer l'évolution de versions d'ontologie. Dans (Chen & Matthews, 2008), les auteurs décrivent une approche d'évaluation des implications sémantiques de changements, basée sur le modèle OWL. Quatre types de métriques ont été définies :

- Des métriques d'évaluation par rapport aux classes. Elles mesurent la stabilité, l'extensibilité et l'obsolescence de classes – entre deux versions d'une ontologie – en considérant respectivement, les classes maintenues, ajoutées et supprimées ;
- Des métriques d'évaluation par rapport aux définitions de classes. Elles mesurent la stabilité, l'extensibilité et l'obsolescence des axiomes sur les définitions des classes entre deux versions d'une ontologie ;
- Des métriques d'évaluation par rapport aux subsomptions de classes. Elles mesurent entre autres, la profondeur moyenne et la profondeur maximale des classes, le nombre de classes généralisées, spécialisées et invariantes entre deux versions d'une ontologie en comparant classe par classe, les variations de profondeur ;
- Des métriques d'évaluation par rapport au modèle logique. Elles mesurent la stabilité, l'extensibilité et l'obsolescence des axiomes logiques– entre deux versions de l'ontologie – en considérant respectivement, les axiomes maintenus, ajoutés et supprimés. Ces métriques sont également appliquées sur les annotations (considérées comme des axiomes non logiques). Le calcul de ces métriques se base sur l'emploi du raisonneur Pellet.

Bien qu'elle soit appliquée pour évaluer l'impact des changements, cette approche ne s'adapte pas au contrôle du processus d'évolution d'une ontologie mais plutôt à l'évaluation de l'évolution de versions d'une ontologie dans le temps. Elle a d'ailleurs été appliquée pour évaluer les évolutions de l'ontologie Gene Ontology[79] sur deux ans (2006-2008).

III- DESCRIPTION DU MODELE DE QUALITE

Dans l'objectif d'optimiser la gestion des changements et de guider le choix entre les différentes résolutions cohérentes proposées – pour un changement – par le processus d'évolution, l'approche *Onto-Evo*^{a}*l* intègre une activité d'évaluation basée sur un modèle de qualité d'ontologies (figure 1). Ce modèle est employé pour évaluer l'impact des différentes résolutions proposées sur la qualité de l'ontologie et ainsi choisir celle qui la préserve.

Pour préserver l'indépendance à l'utilisateur et assurer l'automatisation du processus, la définition du modèle de qualité s'est basée sur des métriques quantifiables, mesurables automatiquement. Le modèle de qualité est décrit sous forme d'un modèle hiérarchique organisé en trois niveaux (figure 2):

- Les *caractéristiques* de qualité correspondant aux aspects : contenu et usage de l'ontologie ;
- Les *critères* de qualité définis pour chaque caractéristique ;

[79] http://www.geneontology.org/

- Les *métriques* de qualité évaluant chaque critère.

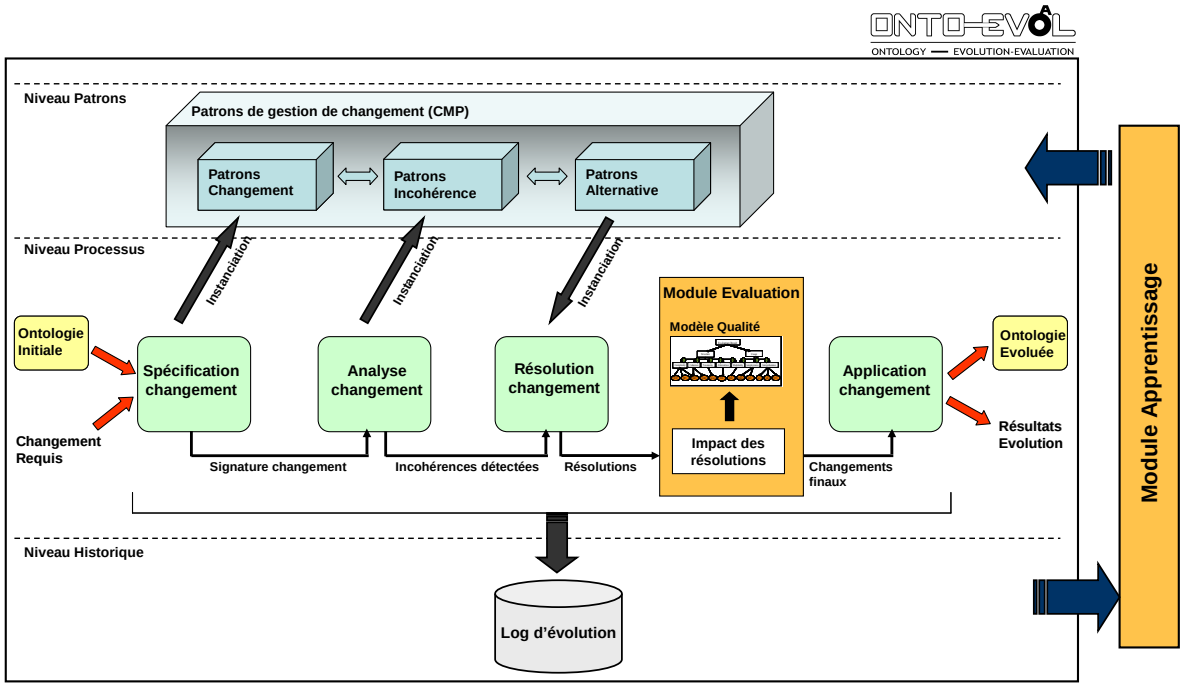


Figure 1. Architecture générale de l'approche *Onto-Evo*^{a}*l.*

Nous avons identifié et défini ces caractéristiques et critères, et sélectionné ces métriques en nous basant sur des résultats de travaux existants que nous avons adaptés et enrichis en fonction de nos besoins et du contexte d'application de l'évaluation dans l'approche *Onto-Evo*^{a}*l.* Nous nous sommes basés particulièrement sur les travaux de (Gangemi et al., 2005) (Gangemi et al., 2006b) (Yang et al. 2006) (Tartir et al., 2005) (Tartir & Arpinar, 2007).

III.1- Caractéristiques de qualité

Les caractéristiques de qualité considérées correspondent aux aspects : *contenu* et *usage* de l'ontologie :

- Le *contenu* de l'ontologie inclut les dimensions structurelle et sémantique de l'ontologie. Son évaluation se base sur les critères de complexité, de cohésion, de conceptualisation et d'abstraction ;
- L'*usage* de l'ontologie tient compte de l'utilisabilité de l'ontologie. Il est évalué par les critères de complétude, et de compréhension.

III.2- Critères de qualité

Les critères considérés par le modèle de qualité sont :

- La *complexité* évaluant les liens structurels et sémantiques entre les entités de l'ontologie, et la navigabilité dans la structure de l'ontologie;
- La *cohésion* tenant compte des composants connectés de l'ontologie (classes et instances) ;
- La *conceptualisation* évaluant la richesse de la conception du contenu de l'ontologie ;

- L'*abstraction* correspondant au niveau d'abstraction des classes (généralisation/spécialisation) en considérant la profondeur des hiérarchies de subsumption;
- La *complétude* considérant le degré de couverture des sources représentatives du domaine modélisé, et de ses propriétés pertinentes en tenant compte de la conformité des désignations (les labels) des classes et des propriétés aux mots clés du domaine ;
- La *compréhension* estimant la facilité de compréhension de l'ontologie à travers le nommage des entités, et les annotations décrivant les définitions de classes et de propriétés, et aussi la facilité d'explication des différenciations entre les classes d'une même hiérarchie (les sous-classes et la classe mère).

Certains critères peuvent être contradictoires : une résolution qui augmente la complexité de l'ontologie par exemple, peut améliorer la cohésion des entités. Pour palier ce problème, nous avons choisi d'attribuer des pondérations aux critères. Ainsi, au début du processus d'évolution, l'expert définit pour chaque critère, une valeur de pondération (w_i), en fonction de son importance pour le domaine modélisé et pour l'application utilisant l'ontologie.

L'évaluation des critères est calculée sur la base de métriques quantifiables (table 1), définies indépendamment du tout langage d'ontologie et de tout domaine ce qui permet notamment, de réutiliser le modèle de qualité pour d'autres contextes d'évaluation d'ontologie.

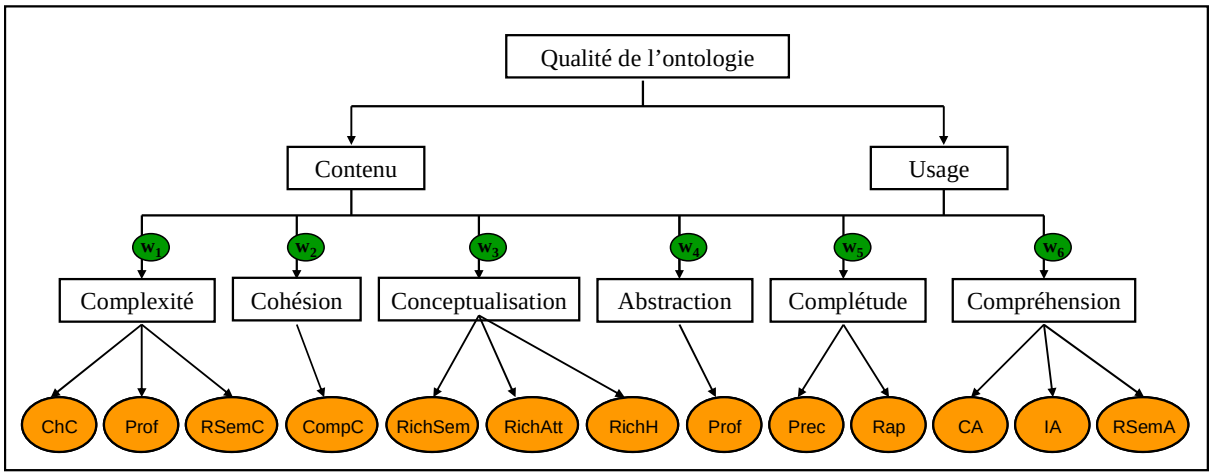


Figure 2. Représentation hiérarchique du modèle de qualité.

III.3- Métriques de qualité

Les solutions de résolution modélisées par les patrons d'alternatives correspondent à :

- Des généralisations/spécialisations de classes qui modifient les subsomptions définies dans l'ontologie et les liens entre les classes et les relations sémantiques (par principe d'héritage de propriété) ;
- Des modifications de domaines et co-domaines de propriétés qui touchent donc aux liens entre les classes et les relations sémantiques ;
- Des définitions de classes hybrides qui augmentent le nombre de classes et de subsomptions dans l'ontologie ;
- Des reclassements d'instances ;
- etc.

Toutes ces alternatives touchent à la structure et la sémantique du contenu de l'ontologie et peuvent avoir un impact sur son usage. Ainsi, en tenant compte des différents patrons d'alternatives définis, nous avons sélectionné, à partir des travaux existants (Gangemi et al., 2005) (Gangemi et al., 2006b) (Yang et al. 2006) (Tartir et al., 2005) (Tartir & Arpinar, 2007), un ensemble de métriques que nous avons adaptées et enrichies pour évaluer l'impact des alternatives de résolutions sur la qualité de l'ontologie. Les différentes métriques évaluant les critères de qualité sont synthétisées par la table 1.

Critères		Métriques
Contenu		
Complexité	(w_1)	- nombre moyen de Chemins pour atteindre une Classe à partir d'une racine de l'ontologie. (ChC)
		- Profondeur moyenne de l'ontologie. (Prof)
		- nombre moyen de Relations Sémantiques par Classe. (RSemC)
Cohésion	(w_2)	- nombre moyen de Composants Connectés (classes et instances). (CompC)
Conceptualisation	(w_3)	- Richesse Sémantique : ratio du nombre total de relations sémantiques assignées aux classes de l'ontologie, divisé par le nombre total de relations dans l'ontologie (relations sémantiques et de subsomption). (RichSem)
		- Richesse des Attributs: ratio du nombre total d'attributs décrivant les classes de l'ontologie, divisé par le nombre total de classes. (RichAtt)
		- Richesse de l'Héritage: nombre moyen de sous-classes par classe. (RichH)
Abstraction	(w_4)	- Profondeur moyenne de l'ontologie. (Prof)
Usage		
Complétude	(w_5)	- Précision: ratio des labels de classes utilisés dans l'ontologie et apparaissant dans une source de référence, par rapport au nombre total des labels de classes de l'ontologie. (Prec)
		- Rappel: ratio des termes utilisés dans une source de référence et apparaissant dans l'ontologie comme labels de classe, par rapport au nombre total de termes dans la source de référence. (Rap)
Compréhension	(w_6)	- ratio du nombre de Classes Annotées, divisé par le nombre total de classes. (CA)
		- ratio du nombre d'Instances Annotées, divisé par le nombre total d'instances. (IA)
		- ratio du nombre de Relations Sémantiques Annotées, divisé par le nombre total de relations sémantiques. (RSemA)

Table 1. Description des métriques de qualité.

Avant de détailler les différentes métriques, nous présentons dans ce qui suit les définitions sur lesquelles se base leur description. Ces définitions étendent le modèle spécifié dans (Tartir et al., 2005) :

- C définit l'ensemble des classes définies dans une ontologie. C_i dénote une classe de C ;
- RSem définit l'ensemble des relations sémantiques définies dans une ontologie. $RSem_i$ dénote une relation sémantique ;
- Att définit l'ensemble des attributs définis dans une ontologie. Att_i dénote attribut ;
- I définit l'ensemble des instances définies dans une ontologie. I_i dénote une instance ;

- C(I) dénote l'instanciation de classe ;
- RSem(I_i,I_j) dénote l'instanciation de relation sémantique ;
- AC définit l'ensemble des annotations de classes définies dans une ontologie. AC_i dénote une annotation de classe ;
- AI définit l'ensemble des annotations d'instances définies dans une ontologie. AI_i dénote une annotation d'instance ;
- ARSem définit l'ensemble des annotations de relations sémantiques définies dans une ontologie. $ARSem_i$ dénote une annotation de relation sémantique ;
- H^{ft} définit l'ensemble des subsomptions de classes. Elle est spécifiée par la fonction d'héritage $H^{ft} \subseteq CxC$. Cette fonction définit une relation directe et transitive entre deux classes: $H^{ft}(C_i, C_j)$ spécifie que C_i est sous-classe de C_j ;
- $RSem^{ft}$ définit l'ensemble des relations sémantiques entre classes. Elle est spécifiée par la fonction $RSem^{ft} \rightarrow CxC$. Cette fonction définit une relation sémantique entre deux classes: $RSem^{ft}(C_i, C_j)$ spécifie les relations sémantiques entre la classe C_i (domaine) et la classe C_j (co-domaine) ;
- dom^{ft} définit l'ensemble des domaines de propriété. Elle est spécifiée par la fonction dom^{ft} : RSem $\rightarrow$ C. Cette fonction définit les domaines d'une relation sémantique : $dom^{ft}(RSem_i)$ donne l'ensemble des classes domaines de $RSem_i$;
- co-dom^{ft} définit l'ensemble des co-domaines de propriété. Elle est spécifiée par la fonction co-dom^{ft} : RSem $\rightarrow$ C. Cette fonction définit les co-domaines d'une relation sémantique : co-$dom^{ft}(RSem_i)$ donne l'ensemble des classes co-domaines de $RSem_i$;
- Att^{ft} définit l'ensemble des attributs de classe. Elle est spécifiée par la fonction Att^{ft} : C $\rightarrow$ Att. Cette fonction définit les attributs d'une classe : Att^{ft} (C_i) donne l'ensemble des attributs décrivant la classe C_i ;
- Ic^{ft} définit l'ensemble des instances de classe. Elle est spécifiée par la fonction Ic^{ft} : C $\rightarrow$ I. Cette fonction définit les instances d'une classe : Ic^{ft} (C_i) donne l'ensemble des instanciations de la classe C_i ;
- Ir^{ft} définit l'ensemble des instances de propriété. Elle est spécifiée par la fonction Ir^{ft} : RSem $\rightarrow$ IxI. Cette fonction définit les instances d'une relation sémantique : Ir^{ft} ($RSem_i$) donne l'ensemble des instanciations de la relation sémantique $RSem_i$;
- H dénote l'ensemble des subsomptions d'une ontologie : H = {(C_i, C_j), avec i $\neq$ j et $H^{ft}(C_i, C_j)$} ;
- H(C_i) dénote l'ensemble des subsomptions d'une sous-hiérarchie de l'ontologie ayant la classe C_i pour racine : H(C_i) : = {(C_j, C_i), avec i $\neq$ j et $H^{ft}(C_j, C_i)$};
- sous-Cls(C_i) dénote l'ensemble des sous-classes d'une classe C_i : sous-Cls(C_i) = {C_j, avec $H^{ft}(C_j, C_i)$};
- superCls(C_i) dénote l'ensemble des superclasses d'une classe C_i : superCls(C_i) = {C_j, avec $H^{ft}(C_i, C_j)$};
- Feuille dénote les C_i vérifiant sous-Cls(C_i) = $\emptyset$. $Feuille_i$ dénote une classe feuille;
- Racine dénote les classes C_i vérifiant superCls(C_i) = $\emptyset$. $Racine_i$ dénote une classe racine ;
- RSemCls(C_i) dénote l'ensemble des relations sémantiques d'une classe C_i : RSemCls(C_i) = {$\cup$ $RSem^{ft}(C_i, C_j)$};
- ClsC dénote l'ensemble des classes C_i vérifiant RSemCls(C_i) $\neq$ $\emptyset$. $ClsC_i$ dénote une classe ayant des relations sémantiques (elle en est le domaine) ;
- RSemIns(I_i) dénote l'ensemble des relations sémantiques d'une instance I_i : RSemIns(I_i) = {$\cup$ $Ir^{ft}(I_i, I_j)$};
- InsC dénote l'ensemble des instances I_i vérifiant RSemIns(I_i) $\neq$ $\emptyset$. $InsC_i$ dénote une instance ayant des relations sémantiques (elle en est l'objet du domaine) ;

- Chemin(C_i, $Racine_i$) définit l'ensemble des chemins d'une classe C_i à une classe racine $Racine_i$. $Chemin_k$(C_i, $Racine_i$) dénote un chemin entre une classe C_i et une classe racine $Racine_i$. Un chemin est une séquence de classes directement connectées par des subsomptions.

III.3.1- Métrique Chemins par Classe (ChC)

Cette métrique définit le nombre moyen de chemins pour atteindre une classe à partir d'une classe racine de l'ontologie. Elle est calculée comme suit :

$$ChC = \Sigma\, j_{[1\,..\,|Racine|]} \left(\Sigma\, i_{[1\,..\,|C|]} \left(|\, Chemin(C_i, Racine_j)\, | \right) / |C| \right) / |Racine|$$

III.3.2- Métrique Profondeur (Prof)

Cette métrique définit la profondeur moyenne de l'ontologie. Elle est calculée comme suit :

$$Prof = \Sigma\, j_{[1\,..\,|Racine|]} \left(\Sigma\, i_{[1\,..\,|Feuille|]} \left(|\, Chemin(Feuille_i, Racine_j)\, | \right) / |Feuille| \right) / |Racine|$$

III.3.3- Métrique Relations Sémantiques par Classe (RSemC)

Cette métrique définit le nombre moyen de relations sémantiques par classe. Elle est calculée comme suit :

$$RSemC = \Sigma\, i_{[1\,..\,|C|]} \left(|\, RSemCls(C_i)\, | \right) / |C|$$

III.3.4- Métrique Composants Connectés (CompC)

Cette métrique définit le nombre moyen de composants connectés dans l'ontologie, c'est-à-dire le nombre moyen de classes et d'instances (y compris celles qui ne sont pas classées à une classe) ayant des relations sémantiques (en considérant les domaines des relations). Elle mesure la connectivité sémantique des classes et des instances. Elle est calculée comme suit :

$$CompC = \left(|\, ClsC\, | + |\, InsC\, | \right) / |C| + |I|$$

III.3.5- Métrique Richesse Sémantique (RichSem)

Cette métrique définit le ratio du nombre total de relations sémantiques assignées aux classes de l'ontologie, divisé par le nombre total de relations dans l'ontologie (relations sémantiques et de subsomption). Elle est calculée comme suit :

$$RichSem = \left(|\, RSem\, | \right) / |\, RSem\, | + |\, H\, |$$

III.3.6- Métrique Richesse des Attributs (RichAtt)

Cette métrique définit le ratio du nombre total d'attributs décrivant les classes de l'ontologie, divisé par le nombre total de classes. Elle est calculée comme suit :

$$RichAtt = \left(|\, Att\, | \right) / |\, C\, |$$

III.3.7- Métrique Richesse de l'Héritage (RichH)

Cette métrique définit le nombre moyen de sous-classes par classe. Elle est calculée comme suit :

$$RichH = \left(\Sigma\, i_{[1\,..\,|C|]} \left(|\ \text{sous-Cls}(C_i)\ |\right)\right) / \left|\ C\ \right|$$

III.3.8- Métrique Précision (Prec)

Cette métrique définit le ratio des labels de classes utilisés dans l'ontologie et apparaissant dans une source de référence, par rapport au nombre total des labels de classes de l'ontologie T_{Onto}. Elle est calculée comme suit :

$$Prec = \left(|T_{Onto} \cap T_{Réf}\ |\right) / \left|\ T_{Onto}\ \right|$$

III.3.9- Métrique Rappel (Rap)

Cette métrique définit le ratio des termes utilisés dans une source de référence et apparaissant dans l'ontologie comme labels de classe, par rapport au nombre total de termes dans la source de référence $T_{Réf}$. Elle est calculée comme suit :

$$Rap = \left(|T_{Onto} \cap T_{Réf}\ |\right) / \left|\ T_{Réf}\ \right|$$

III.3.10- Métrique Classes Annotées (CA)

Cette métrique définit le ratio du nombre de classes annotées, divisé par le nombre total de classes. Elle est calculée comme suit :

$$CA = \left(|\ AC\ |\right) / \left|\ C\ \right|$$

III.3.11- Métrique Instances Annotées (IA)

Cette métrique définit le ratio du nombre d'instances annotées, divisé par le nombre total d'instances. Elle est calculée comme suit :

$$IA = \left(|\ AI\ |\right) / \left|\ I\ \right|$$

III.3.12- Métrique Relations Sémantiques Annotées (RSemA)

Cette métrique définit le ratio du nombre de relations sémantiques annotées, divisé par le nombre total de relations sémantiques. Elle est calculée comme suit :

$$RSemA = \left(|\ ARSem\ |\right) / \left|\ RSem\ \right|$$

IV- RESOLUTION DU CHANGEMENT GUIDEE PAR L'EVALUATION

La phase de résolution du changement de l'approche *Onto-Evo[a]l*, se compose de deux activités : une activité de *proposition de résolutions* basée sur les patrons CMP, permettant de générer des résolutions cohérentes pour le changement à appliquer, et une activité d'*évaluation de résolutions* guidant le choix entre les différentes résolutions proposées en évaluant leur impact sur la qualité de l'ontologie (chapitre 3, figure 2).

L'activité d'évaluation évalue chacune des résolutions sur la base du modèle de qualité :

- Les résolutions sont d'abord, comparées entre-elles métrique par métrique et critère par critère. Tous les critères à l'exception de la complexité, correspondent à un aspect positif de qualité. La comparaison des résolutions s'appuie sur la préservation ou la diminution des valeurs des métriques liées à la complexité, et la préservation ou l'augmentation des valeurs des métriques liées aux autres critères. Les métriques n'ayant pas les mêmes unités, les comparaisons se font métrique par métrique. Pour un même critère, lorsque les valeurs de certaines métriques augmentent et d'autres diminuent, la comparaison se base sur les variations et non les valeurs. Entre les critères, la comparaison des variations tient compte des pondérations des critères afin de sélectionner la résolution qui a le meilleur impact sur les critères de qualité les plus importants ;
- La résolution sélectionnée est alors comparée selon le même principe, métrique par métrique et critère par critère, aux mesures de qualité de la version initiale de l'ontologie (au lancement du processus de gestion de changement). Ces mesures sont enregistrées dans le journal d'évolution. Elles correspondent aux valeurs d'évaluation de l'instance de la classe *Trace_évaluation*, correspondant à l'instance de la classe *Trace_résolution_chgt_sélectionnée*, correspondant à la dernière instance de la classe Trace_évolution dont la valeur de l'attribut *résultat_gestion_évolution* = « validé » (Chapitre 3 , section V.2). Si la résolution préserve (ou améliore) la qualité de l'ontologie, elle sera directement validée et appliquée à la phase d'application du changement du processus. Si par contre, elle a un impact négatif sur la qualité, elle sera présentée à l'utilisateur, avec les résultats des différentes phases du processus, pour décider du changement.

V- DISCUSSION

L'évaluation de qualité d'ontologie a été décrite dans ce chapitre comme une activité essentielle dans le guidage des résolutions de changement. Dans cette section, nous discutons de la place de l'évaluation de qualité par rapport aux différents niveaux de cohérence, gérés par l'approche *Onto-Evo[a]l*, de son apport pour le processus d'évolution, et de l'indépendance du modèle de qualité défini au langage OWL. De plus, nous présentons quelques réflexions sur la prise en compte de la modularité de l'ontologie comme critère de qualité dans l'évaluation des résolutions.

V.1- Evaluation de qualité et cohérence de l'ontologie évoluée

L'approche *Onto-Evo[a]l* gère quatre niveaux de cohérence d'ontologie : la cohérence structurelle, la cohérence logique, la cohérence conceptuelle, et la cohérence de modélisation de domaine (chapitre 3, section VI.1). L'évaluation de la qualité intervient au niveau de :

- La cohérence conceptuelle de l'ontologie en vérifiant la complexité de sa structure, la cohésion de ses composants, la richesse de sa conceptualisation et son niveau d'abstraction ;
- La cohérence de modélisation de domaine en vérifiant sa complétude par rapport à des sources représentatives du domaine.

De plus, elle tient compte de contraintes utilisateurs, liées au domaine de modélisation et/ou au contexte d'application de l'ontologie, à travers les pondérations des critères.

V.2- Apport de l'évaluation de qualité

L'évaluation de la qualité joue un double rôle dans le processus d'évolution :

- Un rôle direct guidant le choix entre plusieurs solutions de résolution d'incohérences ;
- Un rôle indirect puisque l'évaluation de l'impact de l'application – cohérente – du changement sur l'ontologie, en tenant compte de la qualité de l'ontologie initiale (avant évolution), permet d'estimer et justifier le coût du changement.

V.3- Indépendance au modèle OWL

L'évaluation du contenu de l'ontologie s'est focalisée sur l'impact des résolutions sur les classes, les subsomptions, les relations sémantiques, et les liens sémantiques entre classes et entre instances. Le modèle de qualité défini est indépendant de la sémantique du langage OWL DL. Ainsi par exemple, la métrique *relations sémantiques par classe* (RSemC) évaluant la complexité du contenu de l'ontologie, est calculée en se basant sur les définitions de domaine des relations sémantiques (propriétés objets de OWL), sans tenir compte des restrictions de valeur de propriété exprimées sur certaines classes dans le modèle OWL. La sémantique des axiomes OWL du type : ($C_1 \sqsubseteq \forall P.C_2$) ou ($C_1 \sqsubseteq \exists P.C_2$), définit des inclusions de classes (telles que C_1) à des classes anonymes afin de restreindre les relations des individus de ces classes, avec d'autres individus. Ces restrictions portent ainsi sur les instances mais n'affectent pas la complexité du contenu de l'ontologie. Par contre, elles sont prises en compte dans vérification de la cohérence logique de l'ontologie.

En comparaison avec les approches d'évaluation dépendantes du modèle de OWL et intégrant la vérification de la cohérence logique dans leur analyse, dans l'approche *Onto-Evo*[a]*l*, la cohérence logique est vérifiée en amont, à la phase de résolution de changement. Les critères de cohérence logique sont analysés en se basant sur la syntaxe et la sémantique de OWL DL, et les incohérences détectées sont explicitées par les patrons d'incohérences et résolues par des instances de patrons d'alternatives.

V.4- Modularité d'ontologie comme critère de qualité

Une ontologie modulaire est une ontologie composée de sous-ontologies locales formant des modules autonomes dans leur contenu, et inter-reliés, combinés ensemble pour couvrir un large domaine (d'Aquin et al., 2008). La modularité facilite la maintenance et l'enrichissement du contenu de l'ontologie, la réutilisation de l'ontologie et donc, son usage et, le raisonnement sur l'ontologie (contenu et usage). C'est pourquoi, il serait intéressant de considérer la modularité comme critère de qualité et le prendre en compte pour l'évaluation du contenu et de l'usage d'une ontologie. Ainsi, le critère de *modularité* évaluerait la possibilité de décomposer l'ontologie en sous-hiérarchies indépendantes, en calculant le nombre de Modules Distincts dans l'ontologie (MD).

La dimension de modularité est surtout appliquée dans le contexte de larges ontologies ou d'ontologies en réseau[80], où chacune joue le rôle d'un module, et l'ensemble modélise un large domaine. Selon cette optique, un module est lui-même une ontologie dans son contenu et renferme également des informations additionnelles sur la façon de le réutiliser et comment il réutilise d'autres modules (d'Aquin et al., 2008).

Ce n'est pas cette dimension que nous entendons par l'évaluation du nombre du module distincts dans une ontologie. Cette métrique permet juste de tenir compte de la facilité – ou non – d'identifier des sous-

[80] Projet NeOn : http://www.neon-project.org/web-content/

hiérarchies relativement indépendantes dans l'ontologie. Ce critère de modularité assure d'une part, une meilleure gestion de la maintenance du contenu de l'ontologie et d'autres part, sa réutilisation et son adaptation pour des domaines connexes ou des applications différentes.

Le calcul du nombre de modules distincts se baserait ainsi, sur l'identification de sous-hiérarchies de l'ontologie, qui comprennent des classes dépendantes entre-elles (reliées par des subsomptions, des relations sémantiques, des définitions complexes et des instances) et relativement indépendantes des autres sous-hiérarchies de l'ontologie.

Plusieurs techniques de modularisation sont définies dans la littérature (Spaccapietra, 2005) (d'Aquin et al., 2006). Le calcul du nombre de modules dans une ontologie nécessite d'explorer la sémantique des liens entre les classes à travers les subsomptions, les relations sémantiques, mais aussi les axiomes (principe du voisinage axiomatique développé dans le chapitre 5). Il dépend donc du langage de l'ontologie et de la sémantique de son modèle autrement, le calcul sera biaisé.

En effet, limiter le calcul du nombre de modules aux subsomptions et relations sémantiques peut amener à l'identification de modules erronés. Par exemple, nous pouvons délimiter un module dont la racine est une classe C_1 en se basant sur les subsomptions et les relations sémantiques de C_1 et de son voisinage au niveau de la structure. Mais si dans sa description OWL, C_1 est aussi définie comme l'intersection par exemple, de deux classes *A* et *B*, non incluses structurellement dans la sous-hiérarchie de C_1, cet axiome – sans tenir compte de la sémantique des définitions complexes de classes en OWL – sera ignoré dans le calcul du nombre de modules, et les classes *A* et *B* ne seront pas considérées dans le module extrait.

Ainsi, pour intégrer le critère de modularité à l'évaluation, il faudrait adapter le modèle de qualité défini à la sémantique du modèle OWL DL et ainsi, tenir compte des tous les axiomes dans l'identification des modules distincts.

VI- CONCLUSION

Dans ce chapitre, nous avons décrit les différents niveaux de qualité d'ontologie, considérés dans la littérature, et synthétisé les principales approches existantes, notamment celles intégrant l'évaluation de la qualité en évolution d'ontologie. Nous avons ensuite, détaillé l'ensemble des caractéristiques, critères et métriques définissant le modèle de qualité sur lequel se base l'évaluation des résolutions dans l'approche *Onto-Evo^{a}l*, et décrit l'emploi de l'évaluation de qualité pour le guidage de la résolution d'un changement.

Dans le chapitre suivant, nous présentons l'évaluation de l'approche *Onto-Evo^{a}l* et son implémentation par un premier prototype.

Chapitre 7 : Expérimentation et implémentation

Résumé : Dans ce chapitre, nous détaillons l'évaluation de l'approche *Onto-Evo*^{a}*l* que nous avons détaillée dans les chapitres précédents, à travers une première évaluation manuelle basée sur des ontologies test et une deuxième évaluation automatique employant le prototype implémenté. Un modèle pour une architecture complète du système d'évolution d'ontologie est proposé, de même qu'un contexte d'application du système comme plugin d'une plateforme de construction et d'édition d'ontologie est présenté.

I- INTRODUCTION

La gestion de l'évolution d'ontologie est une activité complexe qui nécessite un processus bien structuré assurant une application efficace des changements tout en préservant la cohérence de l'ontologie. La spécification explicite d'un tel processus permet de l'automatiser et de développer un système de gestion d'évolution dont l'efficacité est augmentée par la représentation formelle de la sémantique des changements et la traçabilité des modifications apportées à l'ontologie.

Dans ce chapitre, nous détaillons l'évaluation de l'approche *Onto-Evo*^{a}*l* que nous avons développée dans les chapitres 3, 4, 5 et 6, et nous présentons un premier prototype implémentant un sous-ensemble de ses fonctionnalités.

L'évaluation se base sur une étude empirique appliquant le processus *Onto-Evo*^{a}*l* sur un ensemble d'ontologies test afin de :

- Valider les descriptions des patrons CMP, les relations entre eux, leur applicabilité et l'apport de leur utilisation ;
- Tester le rôle de l'évaluation de qualité dans la comparaison des résolutions et la sélection de celle qui préserve la qualité de l'ontologie évoluée ;
- Vérifier l'application cohérente d'un changement final validé (changement requis et les changements de résolution dérivés) ;
- S'assurer de la traçabilité de l'évolution de l'ontologie et des résultats de gestion de changements.

Une première évaluation a été conduite manuellement sur un ensemble d'ontologies disponibles sur le Web. Pour chaque ontologie test, nous avons délimité une sous-hiérarchie que nous ne considérons pas dans l'ontologie de départ, et de laquelle nous extrayons les changements à ajouter. Chaque changement est spécifié par un patron de changement de l'ontologie CMP. Ensuite, son analyse et la proposition des résolutions sont menées étape par étape, en suivant les descriptions des patrons CMP correspondants, et les relations entre eux. La manipulation des ontologies test et la vérification des incohérences se sont basées sur l'utilisation de l'outil Protégé[81] et du raisonneur Pellet[82].

[81] http://protege.stanford.edu/
[82] http://clarkparsia.com/pellet/

La deuxième évaluation a été conduite automatiquement. Elle s'est basée sur des expérimentations appliquées en exécutant le prototype implémenté. Le prototype a été développé au fur et à mesure de la spécification de l'approche *Onto-Evo^{a}l*. Celle-ci ayant d'abord été focalisée sur la résolution automatisée d'une incohérence logique et le guidage du choix de résolution par intégration d'une activité d'évaluation de qualité, ce sont ces deux fonctionnalités qui ont été implémentées et testées de manière plus complète. Ensuite, à travers les différentes applications, nous avons identifié des invariants dans les types d'incohérences détectées et les types de résolution adaptées, ce qui nous a conduits à identifier et spécifier des patrons de gestion de changements : les patrons CMP. La version actuelle du prototype supporte un sous ensemble minimal de patrons de changements basiques, pour lesquels un sous ensemble de patrons d'incohérences et d'alternatives, sont associés.

Ainsi, dans la section 2 du chapitre, nous exposons les critères considérés pour l'évaluation de l'approche. La description de l'évaluation manuelle par des ontologies test et les observations retenues de cette évaluation sont présentées dans la section 3. Dans la section 4, nous présentons l'évaluation automatique de l'approche *Onto-Evo^{a}l* à travers le prototype implémentant un ensemble de ses fonctionnalités. Un prototypage de l'architecture du système implémentant l'approche globale est proposé dans la section 5. Le contexte d'une application concrète du système dans une plateforme de construction et d'édition d'ontologie, est décrit dans la section 6. La synthèse des différents points discutés dans le chapitre est présentée en conclusion dans la section 7.

II- CRITERES D'EVALUATION

Les critères considérés dans l'évaluation l'approche *Onto-Evo^{a}l* sont classés en quatre niveaux :

- Niveau patrons CMP :
    - Facilité de spécification d'un changement par un patron de changement,
    - Prévision des incohérences potentielles attendues à partir des contraintes spécifiées dans les patrons de changements instanciés,
    - Explicitation des incohérences à l'aide des patrons d'incohérences,
    - Génération des patrons d'alternatives,
    - Applicabilité des instances d'alternatives générées,
    - Constitution de résolutions globales cohérentes.
- Niveau qualité de l'ontologie :
    - Rôle de l'évaluation de qualité dans le guidage du choix de la résolution préservant la qualité. A ce niveau, nous évaluons l'apport de l'évaluation de qualité et non le modèle de qualité lui-même. L'aspect de qualité évalué est le contenu de l'ontologie puisque l'évaluation de l'usage suppose de disposer de sources de référence pour chaque domaine modélisé. De même, n'ayant pas pris en compte la maintenance des annotations dans le processus de gestion, nous n'avons pas tenu compte du critère de compréhension de l'ontologie.
- Niveau ontologie évoluée :
    - Application cohérente du changement final (changement requis et les changements dérivés).
- Niveau journal d'évolution :
    - Traçabilité des résultats de gestion pour une transaction de changement et traçabilité de l'évolution de l'ontologie suivie sur des applications successives de changements.

III- EVALUATION MANUELLE

Pour valider les patrons CMP définis, les liens conceptuels entre les trois types de patrons (patrons de changements, patrons d'incohérences et patrons d'alternatives) et l'intégration d'une modélisation par patron dans un processus d'évolution, nous avons procédé à une évaluation manuelle sur des ontologies test, en appliquant les patrons de changements spécifiés par l'ontologie CMP, à chaque ajout d'un changement.

Pour assurer la cohérence des changements ajoutés par rapport au domaine modélisé par une ontologie test, nous avons délimité, pour chaque ontologie test, une sous-hiérarchie que nous ne considérons pas dans l'ontologie de départ, et de laquelle nous extrayons les changements à ajouter. Chaque changement est spécifié par un patron de changement de l'ontologie CMP.

III.1- Sélection des ontologies test

Pour la sélection des ontologies test, nous nous sommes basés sur un ensemble d'ontologies mises à disponibilité des utilisateurs, notamment pour des besoins d'étude et de test. Certaines de ces ontologies ont été collectées à partir du moteur de recherche Web sémantique *Watson*[83], l'annuaire d'ontologies *TONES*[84] de l'Université de Manchester et du site de FAO[85] (Food and Agriculture Organization for the United Nations).

Pour les besoins de l'expérimentation, nous avons cherché à avoir des ontologies de taille relativement petite (< 100 primitives) afin de faciliter la manipulation des opérations de changements test et l'application des patrons, tout en sélectionnant des ontologies ayant une TBox suffisamment riche en classes, en propriétés objet (relations sémantiques) et propriétés de données (attributs), en subsomptions, et en axiomes (disjonctions, restrictions de valeurs, définitions complexes de classes telles que les intersections de classes, etc.), et aussi, une ABox contenant des instanciations de classes et de propriétés. Afin de varier l'échantillon test et répondre à ces besoins, nous avons sélectionné certaines ontologies riches de taille moyenne (> 100 primitives), desquelles nous avons considéré des sous-parties (< 100 primitives).

L'ontologie des vins *Wine* est très riche en axiomes OWL DL et a contribué essentiellement à la vérification des patrons d'incohérences et d'alternatives ainsi qu'à la définition de classes complexes. L'ontologie *Pizza* définie et mise à disposition par l'Université de Stanford est une ontologie assez complète en type d'axiomes pour être utilisée dans les tutoriaux sur *Protégé*. Nous avons également considéré l'ontologie de géopolitique et des ontologies du domaine de l'agriculture (ontologie des plantes sauvages et ontologie pêche)[86] référencées sur le site de FAO. De même, nous avons considéré une ontologie de tourisme *Travel* minimale mais avec une TBox et une ABox riches, une ontologie médicale de pneumologie *OntoPneumo* (Baneyx et al., 2005) fournie par l'INSERM[87], et une ontologie de modèles e-business *eBMO* (Osterwalder, 2004). Ayant travaillé sur cette ontologie (Djedidi, 2005), nous l'avons employée pour tester le processus dans une optique de construction incrémentale. L'ontologie eBMO définit principalement des subsomptions et des propriétés, et a permis de valider les patrons de changements basiques liés aux subsomptions de classes et de propriétés et à la définition de domaines et co-domaines.

[83] http://watson.kmi.open.ac.uk
[84] http://owl.cs.manchester.ac.uk/repository/
[85] http://www.fao.org/
[86] http://aims.fao.org/fr/website/Domain%20Ontologies/sub
[87] http://www.inserm.fr/fr/

La description des ontologies considérées est synthétisée par le tableau ci-après (table 1):

Ontologie	Classes	Subsomptions	Relations sémantiques	Attributs	Instances	Disjonction	Restriction	Classes complexes
Wine	74	134	13	1	123	oui	oui	Oui
Pizza	81	92	8	0	5	oui	oui	Oui
Geopolitical (FAO)	12	10	6	40	290	oui	oui	Non
Onto plantes sauvages (CWR)	> 100	> 100	67	10	0	Non	Oui	Non
Onto Pêche FOS	> 100	> 100	> 100	22	> 1000	oui	Oui	Oui
Travel	35	29	6	4	3	Oui	Oui	Oui
Onto Pneumo	> 100	> 100	11	0	0	Oui	Oui	oui
eBMO	24	22	18	26	0	non	non	non

Table 1. Description des ontologies test.

III.2- Patrons CMP évalués

La liste des patrons CMP testés est synthétisée par le tableau suivant (table 1) :

Patrons de changements basiques	
Classe de patrons	**Liste des patrons**
Classe	Ajouter (classe, union de classes, intersection de classes, complément de classes, énumération), Etendre définition (par une union de classes, une intersection de classes, un complément de classes, une énumération), Supprimer
Sous-classe	Ajouter, Supprimer
Equivalence de classe	Ajouter (à une classe, une union de classes, une intersection de classes, un complément de classes, une énumération), Supprimer
Disjonction de classe	Ajouter (à une classe, une union de classes, une intersection de classes, un complément de classes, une énumération), Supprimer
Restriction de valeur	Ajouter (égalité, universelle, existentielle), Supprimer
Restriction de cardinalité	Ajouter (maximale, minimale, exacte), Supprimer
Propriété	Ajouter, Supprimer
Domaine de propriété	Ajouter (classe, union de classes, intersection de classes, complément de classes, énumération, restriction de valeur, restriction de cardinalité), Modifier, Supprimer
Co-domaine de propriété	Ajouter (propriété objet (union de classes, intersection de classes), propriété de données), Modifier (propriété objet, propriété de données), Supprimer (propriété objet, propriété de données)

Sous-Propriété	Ajouter, Supprimer
Propriété fonctionnelle	Mettre (propriété objet), Annuler
Propriété inverse fonctionnelle	Mettre, Annuler
Propriété inverse	Ajouter, Supprimer
Propriété symétrique	Mettre, Annuler
Propriété transitive	Mettre, Annuler
Individu	Ajouter, Etendre instanciation (classe, domaine ou co-domaine de propriété), Supprimer, Supprimer instanciation (classe, domaine ou co-domaine de propriété)
Patrons de changements complexes	
Sous-hiérarchie	Ajouter une classe et ses sous-classes.
Multi-héritage	Ajouter une sous-classe en multi-héritage.
Fusion	Fusionner deux classes.
Généralisation	Remonter une classe.
Spécialisation	Descendre une classe.
Suppression	Supprimer une classe en rattachant ses propriétés à ses sous-classes, et ses instances et sous-classes à une superclasse.
Domaine de propriété	Etendre le domaine d'une propriété à une superclasse, Restreindre le domaine d'une propriété à une sous-classe.
Co-domaine de propriété	Etendre le co-domaine d'une propriété à une superclasse, Restreindre le co-domaine d'une propriété à une sous-classe.
Instanciation	Etendre l'instanciation d'un individu multiple (ajouter plusieurs instanciations à un individu existant).
Agréger propriétés	Agréger propriétés de deux classes.
Restriction de valeur complexe (union, intersection ou complément de classes)	Restriction de valeur universelle complexe, Restriction de valeur existentielle complexe, Restriction de valeur d'égalité complexe.
Patrons d'incohérences	
Disjonction	Subsomption, Instanciation, Ascendants
Restriction de valeur	Restriction universelle inapplicable, Incompatibilité de restrictions universelle et existentielle
Suppression d'une classe	Dépendances obsolètes (propriétés, sous-classes et instances)
Définition complexe de domaine ou co-domaine	Intersection de domaines disjoints, Intersection de co-domaines disjoints
Patrons d'alternatives	
Résolution d'incohérences de disjonction liée à la subsomption par classe hybride	Définir une classe hybride pour une subsomption, Relier toutes les subsomptions à une classe hybride.
Résolution d'incohérences de disjonction liée à l'instanciation par classe hybride	Définir une classe hybride pour une instance, Relier toutes les instances à une classe hybride.
Résolution d'incohérences de disjonction basée sur les définitions de classe	Etendre définition de toutes les subsomptions, Etendre définition d'une sous-classe, Etendre définition d'une sous-classe par rapport aux ascendants disjoints.

Résolution d'incohérences de subsomption par généralisation	Définir une subsomption généralisée.
Résolution d'incohérences de restriction de valeur	Résolution de restriction universelle inapplicable, Résolution d'incompatibilité de restrictions universelle-existentielle, Résolution d'incompatibilité de restrictions existentielle-universelle.
Résolution d'incohérences de dépendances obsolètes	Détacher les dépendances obsolètes, Reclasser et Redistribuer les dépendances obsolètes
Résolution d'intersection de domaines ou co-domaines disjoints	Transformer domaine intersection en domaine union, Transformer co-domaine intersection en co-domaine union

Table 2. Synthèse des patrons CMP évalués.

III.3- Détails de l'évaluation manuelle

L'objectif de cette expérimentation est d'évaluer l'apport des patrons CMP modélisés pour la gestion des changements d'ontologie à travers une étude empirique sur la capacité d'appliquer les patrons CMP définis pour spécifier des changements, expliciter des incohérences et proposer des alternatives applicables pour les résoudre ; sur la possibilité de combiner des alternatives pour constituer des résolutions globales applicables ; sur la cohérence des résolutions globales constituées ; sur l'apport de l'évaluation de la qualité dans la sélection de la résolution à retenir ; et sur la validité d'un changement finalisé.

Pour chaque ontologie test considérée, nous avons tenté d'isoler une sous-hiérarchie riche en certains types d'axiomes (des subsomptions, des définitions complexes, des restrictions, …) et/ou de primitives (attributs, relations sémantiques, individus, classes). Certes, la sous-hiérarchie isolée d'une ontologie test, étant extraite de cette ontologie, ne causera pas d'incohérences si elle est ajoutée en un seul lot. Mais, ajouter les axiomes contenus dans cette sous-hiérarchie un à un, pas forcément dans le même ordre de leur création dans l'ontologie initiale (ontologie test avant extraction de la sous-hiérarchie), peut altérer la cohérence de l'ontologie à modifier (ontologie test après extraction de la sous-hiérarchie). Notons qu'au début des évaluations, nous nous sommes assurés que l'ontologie test ainsi que l'ontologie à modifier (ontologie test après extraction de la sous-hiérarchie), sont cohérentes.

La manipulation des ontologies test et la vérification des incohérences se sont basées sur l'utilisation de l'outil Protégé et du raisonneur Pellet.

Première partie de l'évaluation

Pour chaque axiome extrait d'une sous-hiérarchie source de changements, d'une ontologie test, nous avons procédé comme suit :

- Nous identifions à partir du type du changement le patron de changement correspondant, et nous vérifions que la description du patron permet bien de le spécifier ;
- Nous évaluons ensuite la possibilité de prévoir les incohérences attendues de l'application de ce changement à partir des contraintes exprimées dans le patron ;
- Nous vérifions ensuite, le type des incohérences détectées et leur correspondance avec patrons d'incohérences. Nous examinons si les patrons d'incohérences appariés aux incohérences détectées sont associés ou non au patron de changement étudié. Et nous testons l'instanciabilité du patron d'incohérences sélectionné par rapport au contexte de localisation de l'incohérence ;

- Nous vérifions également, la possibilité d'expliciter les incohérences instanciées aux patrons d'incohérences par la description de ces patrons ;
- Nous évaluons la possibilité de générer à partir des relations de résolution entre les patrons CMP instanciés, des instances d'alternatives applicables ;

Notons que pour les changements de type suppression de classes, de propriétés, ou d'instances, nous avons opéré sur l'ontologie test directement.

Deuxième partie de l'évaluation

Dans cette partie de l'évaluation, nous nous intéressons au cas de changement ayant aboutit à l'identification des incohérences détectées et la proposition d'alternatives applicables. L'objectif est de tester la possibilité de constituer, à partir de la combinaison des alternatives proposées pour chaque incohérence, des résolutions globales du changement, qui soient applicables ; et d'estimer le nombre de résolutions cohérentes par rapport à toutes les résolutions globales dérivées.

Troisième partie de l'évaluation

Dans cette partie de l'évaluation, nous nous intéressons au test de l'instanciation des différentes classes historique dans l'ontologie journal d'évolution pour sauvegarder les résultats de traitement d'un changement. A chacune des étapes d'avancement décrites dans les parties précédentes de l'évaluation, nous avons instancié manuellement les classes correspondantes dans l'ontologie journal pour valider sa conceptualisation et s'assurer de la traçabilité des résultats de gestion de changement.

III.4- Observations retenues de l'évaluation manuelle

L'objectif de cette expérimentation est d'évaluer les critères relatifs aux niveaux *patrons* et *journal d'évolution* pour s'assurer de l'utilisabilité des patrons CMP et de leur contribution au contrôle et au guidage de la gestion d'un changement, et aussi de la possibilité de tracer les différents résultats de gestion en instanciant les classes de l'ontologie journal.

Les observations retenues de l'évaluation manuelle sont les suivantes :

- Tous les patrons de changements basiques et composés sont instanciables ;
- La plupart des contraintes non vérifiées des patrons de changements instanciés (vérification de l'axiome = *false*) ont permis de prévoir les incohérences attendues. La correspondance des incohérences attendues aux incohérences détectées, a surtout été observée sur des changements appliqués à des classes (et des propriétés) instanciées par des individus ;
- La majorité des incohérences détectées correspondent à des patrons d'incohérence modélisés. Une fois identifiées par le raisonneur, leur explicitation par la description du patron correspondant est facilitée. Néanmoins, sur certains cas, la distinction entre les axiomes concernés par l'incohérence et ceux responsables n'est pas toujours claires. Ceci est en partie lié aux limites d'une évaluation manuelle, car le raisonneur ne suffit pas pour diagnostiquer l'incohérence ;
- Nous avons remarqué que les incohérences détectées qui n'ont pas pu être identifiées faute d'une localisation précise et automatique, découlent pour la plupart de l'application de changements complexes et de la manipulation de classes possédant plusieurs instances ;
- L'ensemble des incohérences identifiées à partir des liens entre les patrons CMP, incluent des incohérences non détectées par le raisonneur, essentiellement lorsque les classes n'ont pas d'instances. Cette observation a été notée sur les changements d'*extension de domaine de propriété par des intersections de classes* et des changements de *définition de restrictions de valeur*. A travers les contraintes d'application de ces types de changement, définies dans les

descriptions de leurs patrons respectifs, les incohérences identifiées par la modélisation des CMP, permettent d'éviter des insatisfactions implicites de classes qui seront dévoilées ultérieurement à l'instanciation des classes, concernées par ces types de changement. L'instanciation des patrons correspondant à ces incohérences implicites (non détectées) permet d'anticiper leur résolution et d'éviter des incohérences potentielles qui ne seront révélées qu'à la modification de la ABox (changements d'instanciation) et qui seront plus difficiles à résoudre plus tard. Ces observations ont permis de mettre en avant les incohérences supportées (identifiées et résolues) par l'approche *Onto-Evo^{a}l* qui sont difficilement appréhendables par l'utilisateur. D'autant plus que l'utilisateur n'est pas forcément un ontologue expérimenté. La gestion contrôlée et argumentée des changements permet d'assister tout utilisateur d'un outil d'édition intégrant le système de gestion d'évolution ;

- Les instances d'alternatives définies à partir des patrons d'alternatives générés, sont pour la plupart applicables c'est-à-dire, qu'elles s'adaptent à leur contexte d'application (identification de toutes entités de l'ontologie de domaine référencées par les axiomes de résolution) ;
- Les résolutions globales constituées de la combinaison des alternatives proposées pour chaque incohérence, n'ont pas toujours été intuitives, particulièrement pour les ontologies riches en axiomes. La difficulté a surtout été observée sur des changements complexes ;
- Sur les résolutions bien constituées, 50% en moyenne sont cohérentes. Nous avons alors vérifié l'interprétabilité des incohérences causées par les résolutions incohérentes, par les patrons d'incohérences, et réussi à identifier les plus simples (celles qui n'impliquent pas plusieurs entités et qui concernent des classes peu instanciées et avec un nombre limité d'axiomes). Les conditions d'une évaluation manuelle ne permettent pas de pousser plus loin l'analyse de ces incohérences ;
- Sur un changement basique *d'ajout d'une relation de sous-classe incohérente* (incohérence de disjonction entre les deux superclasses) et un changement complexe cohérent d'ajout d'une sous-hiérarchie, nous avons validé par l'instanciation manuelle des classes historique de l'ontologie journal, la conceptualisation des classes traçant la gestion du changement de la spécification à la génération de résolutions globales cohérentes.

L'évaluation des autres critères considérés dans cette étude empirique a été menée à travers une évaluation automatique que nous détaillons dans la section suivante.

IV- EVALUATION AUTOMATIQUE

L'évaluation automatique se base sur le prototype que nous avons développé, supportant la gestion d'un premier sous-ensemble de patrons de changements basiques et implémentant un sous-ensemble des fonctionnalités de l'approche.

IV.1- Description du prototype

Le prototype de gestion de changement a été développé en langage Java. Son implémentation s'est basée sur l'utilisation de OWLAPI[88], une API standard *open source* pour la modélisation, la construction et l'interrogation d'ontologies (Horridge et al., 2007). Le prototype intègre le raisonneur Pellet dans sa version 2.0.0 à travers lequel il peut manipuler la TBox et la ABox d'ontologie OWL DL.

Le prototype implémente une version minimaliste du processus de gestion de changement *Onto-Evo^{a}l*. Ayant été implémentée au fur et à mesure de la spécification de l'approche, et bien avant l'identification de classes de résolutions types qui nous a conduits à modéliser les patrons CMP, sa version 0 supporte la

[88] http://owlapi.sourceforge.net/

gestion contrôlée d'un ensemble de changements basiques, de la spécification d'une demande de changement à partir d'une interface descriptive, à la validation de son application. Elle assure la détection des incohérences (en faisant appel au raisonneur Pellet), et la résolution des incohérences de disjonctions liées à la subsomption et à l'instanciation, et des incohérences de dépendances obsolètes causées par la suppression d'une classe. Une seule incohérence par changement est supportée, par contre, plusieurs alternatives peuvent être proposées. Le module d'évaluation de qualité qui y est intégré, permet de guider le choix entre les différentes alternatives en considérant leur impact sur la qualité de l'ontologie. La comparaison des mesures de qualité entre l'ontologie avant, et, après l'application d'un changement – avec une alternative de résolution – tient compte des pondérations définies par l'utilisateur.

La version actuelle du prototype 1.0 (version α) reprend les changements, les incohérences et les alternatives supportés par l'ancienne version en manipulant leur modélisation par les patrons CMP correspondants.

IV.2- Description de l'évaluation automatique

En complément à l'évaluation manuelle, l'objectif de cette expérimentation est de tester l'automatisation du processus pour un sous ensemble de changements basiques et d'évaluer les critères relatifs à la *qualité du contenu de l'ontologie évoluée.*

Toujours en utilisant le même ensemble d'ontologies test sélectionné dans l'évaluation manuelle, nous avons d'abord évalué la gestion automatisée de changements basiques, en focalisant les tests sur la localisation et la justification des incohérences détectées (disjonctions et dépendances obsolètes) et la résolution cohérente des changements. La résolution des incohérences testées concorde avec les propositions générées par les patrons d'alternatives et fournit des alternatives cohérentes.

Gérant une seule incohérence par changement, nous avons ensuite évalué l'apport de l'évaluation de la qualité pour la sélection de l'alternative à appliquer. Seuls les critères et métriques d'évaluation de la qualité du contenu des ontologies ont été considérés. Les critères et métriques évaluant l'usage supposent la disponibilité de source(s) de référence représentative(s) du domaine. De même, n'étant pas prises en compte dans la gestion des changements, les annotations des entités n'affectent pas l'évaluation de l'impact des alternatives.

Les alternatives sont comparées entre elles et aux anciennes valeurs de qualité de l'ontologie initiale (calculées et sauvegardées dans le journal d'évolution). Pour une seule alternative cohérente proposée, l'évaluation de la qualité permet de tenir compte de l'impact du changement final (le changement de départ et l'alternative dérivée) sur la qualité de l'ontologie et de notifier l'utilisateur sans valider le changement si la qualité est altérée. Lorsqu'il y a plus d'une alternative, et bien que les changements soient basiques, l'évaluation de la qualité montre – en fonction de la taille de l'ontologie test considérée, du nombre de changements dérivés inclus dans les alternatives (permettant de les réaliser) et en fonction des pondérations affectées aux critères – une variation comparable entre les impacts des différentes alternatives. Ce qui permet de choisir entre les résolutions tout en vérifiant par la suite, que la résolution sélectionnée préserve la qualité de l'ontologie avant le changement.

Ces observations ont été confirmées essentiellement par des changements de suppression de classes possédant plusieurs sous-classes et instances, et référencées comme domaine (ou co-domaine) de plusieurs propriétés ce qui cause des dépendances obsolètes. Les variations de l'impact sur la qualité entre une alternative al_1 qui {supprime les axiomes définissant la classe à supprimer comme domaine (ou co-domaine), supprime les relations de subsomptions sur cette classe, et supprime les relations d'instanciations de cette classe}, et une alternative al_2 qui {restreint les domaines (ou co-domaine) des propriétés référençant la classe à supprimer à ses sous-classes, reclasse ses sous-classes en les attachant à ses superclasses, et redistribue ses instances en les affectant à ses superclasses} sont notables ; et le choix

entre al_1 ou al_2 est clairement justifié par ces variations selon que les pondérations des critères priorisaient la complexité (al_1) ou la cohésion, la conceptualisation et l'abstraction (al_2) du contenu de l'ontologie.

L'application finale d'un changement résolu – préservant la qualité – est ensuite validée pour confirmer la cohérence de l'ontologie évoluée. De plus, pour compléter l'évaluation des critères du niveau *journal d'évolution*, nous avons testé la sauvegarde – manuelle – des résultats d'évaluation de l'impact des résolutions sur la qualité, et de la validation finale d'une évolution d'ontologie.

V- PROTOTYPAGE DE L'ARCHITECTURE GLOBALE DU SYSTEME ONTO-EVO^{A}L

Dans sa version actuelle, l'implémentation de l'approche *Onto-Evo^{a}l* correspond à un premier prototype qui est loin d'être un système complet d'évolution d'ontologie explorant la sémantique apportée par la modélisation des patrons CMP, et la richesse des connaissances sauvegardées dans le journal d'évolution. Le prototype a besoin d'être complété pour permettre l'application de l'ensemble des patrons définis, et enrichi par un ensemble de composants logiciels implémentant les détails techniques des différentes phases du processus.

Dans cette section, nous avons voulu proposer une architecture globale pour le système *Onto-Evo^{a}l*. L'architecture est détaillée par rapport aux composants logiciels dans la figure 1 et par rapport aux artefacts produits dans la figure 2.

Les principaux composants logiciels sont :

- Le *descripteur de patron* qui permet d'extraire une sous-hiérarchie de l'ontologie CMP correspondant à un patron, et de décrire en axiomes l'interface correspondante (la partie instanciable) ;
- Le *sélectionneur de patron* qui vérifie la mise en correspondance d'un patron de changement à un changement demandé, ou un patron d'incohérence à une à une incohérence causée ;
- Le *générateur de patron* qui assure la génération des patrons d'alternatives à partir des relations de résolution avec les patrons de changements et d'incohérences ;
- L'*instanciateur de patron* dont le rôle est d'instancier un patron CMP et l'adapter au contexte de son application ;
- Le *spécificateur de changement* permet de formaliser une demande de changement selon une instance de patron de changement ;
- Le *simulateur de changement* permet d'appliquer temporairement le changement à gérer sur une copie de l'ontologie pour le traiter ;
- Le *localiseur d'incohérence* permet de localiser et diagnostiquer une incohérence détectée ;
- Le *générateur de résolution globale* permet de combiner les différentes alternatives proposées pour les incohérences causées et constituer toutes les résolutions globales possibles ;
- L'*évaluateur de qualité* évalue l'impact de toutes les résolutions cohérentes générées sur la qualité de l'ontologie en tenant compte du modèle de qualité, des pondérations des critères et de la qualité de l'ontologie initiale (avant le changement) ;
- Le *validateur de changement* permet de décider de l'application du changement en fonction des résultats de l'évaluation ;
- L'*archiveur* sauvegarde les traces de tous les résultats de traitement de changement et des évolutions de l'ontologie ;
- Le *module d'apprentissage* a pour objectif d'exploiter les connaissances issues de la base de connaissances du journal d'évolution pour enrichir l'ontologie des patrons CMP.

La gestion d'une transaction de changement produit des artefacts à deux niveaux : le niveau processus et le niveau historique (figure 2). Les artefacts du niveau processus se composent de la signature du

changement demandé décrivant une instance de patron de changement, les instances de patrons d'incohérences décrivant les incohérences causées, des instances de patrons d'alternatives décrivant les alternatives générées, les résolutions globales constituées de la combinaison des différentes alternatives, les résultats d'évaluation de l'impact des résolutions globales cohérentes sur la qualité, et les résultats de la validation du changement. Tous ces artefacts sont enregistrés sous forme d'une instance de classe historique dans le journal d'évolution (Chapitre 3, section 5).

Le journal d'évolution constitue une véritable base de connaissances, il comprend les entités ontologiques décrivant l'historique d'évolution et les instances correspondantes aux traces d'exécution du processus, d'évolution d'une version de l'ontologie, et celles des différentes versions d'une ontologie.

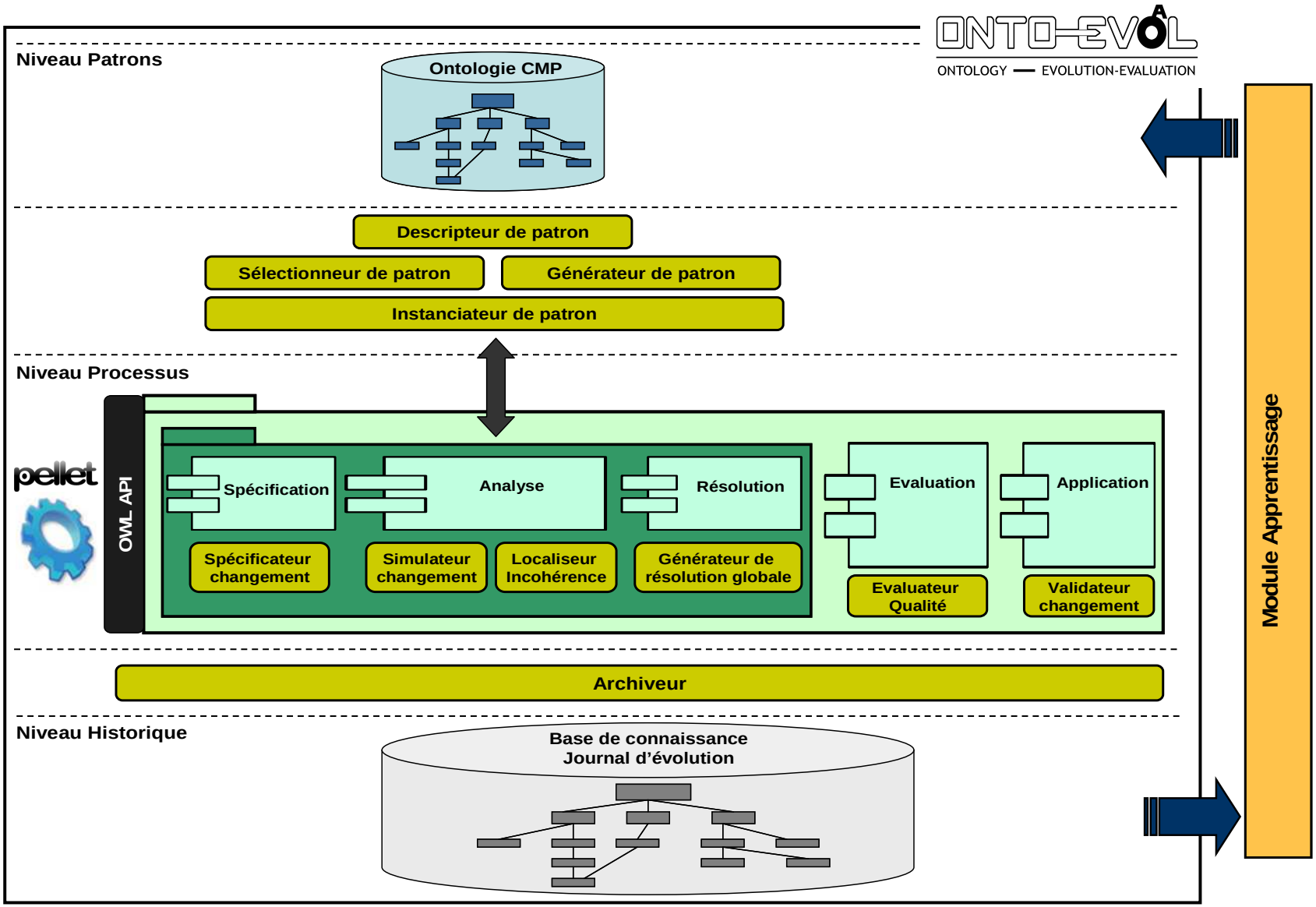


Figure 1. Architecture globale par rapport aux composants logiciels.

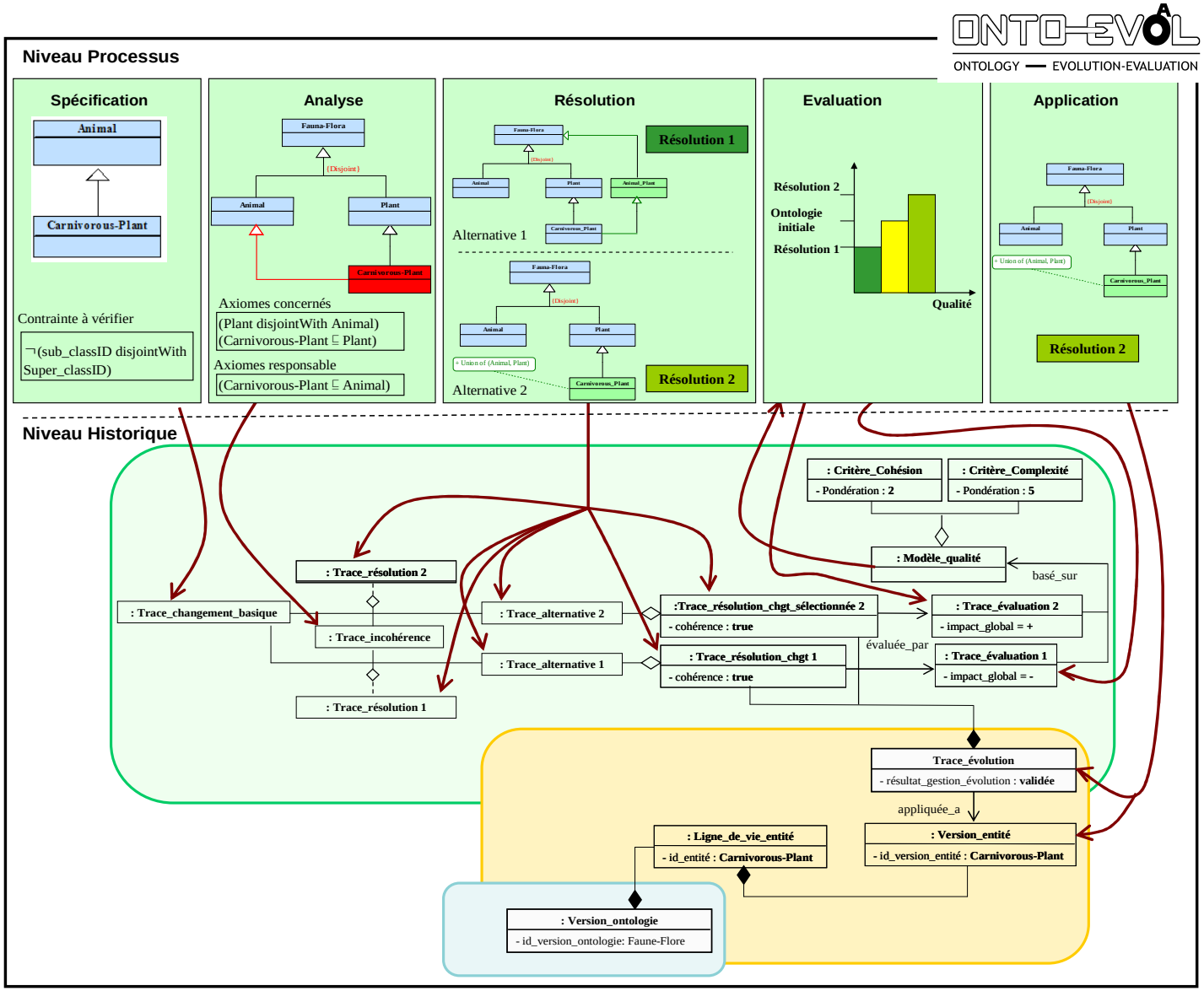


Figure 2. Architecture globale par rapport aux artefacts produits.

VI- INTEGRATION DU SYSTEME D'EVOLUTION A UNE PLATEFORME DE CONSTRUCTION ET D'EDITION D'ONTOLOGIE : DAFOE

Les travaux de recherche menés dans le cadre de cette thèse ont été intégrés dans le projet de recherche RNTL DAFOE4App[89] (Differential And Formal Ontology Editor for Applications) développant une plateforme de construction et d'édition d'ontologies. L'objectif du projet est de proposer une méthode complète associée à une plateforme technique pour concevoir des ontologies, de la modélisation à partir du domaine à leur évolution en passant par leur formalisation et exploitation. La plateforme est organisée en quatre couches, chacune permettant de considérer des ressources plus ou moins formelles (textes, termes, concepts et primitives ontologiques formelles). Selon les ressources en entrée, différents niveaux de sortie correspondant à des produits de plus en plus élaborés sont prévus (Charlet et al., 2009): (1) des réseaux terminologiques, (2) des concepts termino-ontologiques, et (3) des entités ontologiques formelles. Le cadre méthodologique de la plateforme DAFOE est décrit dans (Charlet et al., 2008).

VI.1- Système d'évolution comme plugin de la plateforme DAFOE

L'idée est de greffer le système d'évolution d'ontologie comme un plugin à la plateforme DAFOE. Le plugin assurera la gestion d'évolution d'ontologie au niveau de la couche formelle de la plateforme (figure 3). Le processus de gestion de changement est lancé suite à une demande d'application d'un changement au niveau de l'interface d'édition d'ontologies formelles. A la fin de la gestion du changement demandé, le plugin retourne la confirmation de l'application du changement en le faisant apparaitre dans l'arborescence de l'ontologie avec l'ensemble des changements dérivés et affiche une synthèse des résultats de gestion du changement : les incohérences détectées, la résolution globale retenue et appliquée et son impact sur la qualité. Si le changement ne peut pas être appliqué, tous les résultats de gestion de changement en fonction du point d'arrêt du processus, sont communiqués à l'utilisateur pour décider du changement : les résultats de la spécification du changement, la prévision des incohérences attendues, l'analyse des incohérences détectées, leur localisation et explicitation par les patrons d'incohérences, la génération des alternatives potentielles, la constitution des résolutions globales et l'évaluation des résolutions. Ces résultats sont extraits des traces d'exécution du processus, enregistrées dans le journal d'évolution.

Les besoins fonctionnels du plugin d'évolution d'ontologie formelle sont détaillés dans la section suivante.

VI.2- Besoins fonctionnels du plugin d'évolution

Le plugin opère en interne et sans interactivité avec l'utilisateur à part pour la communication des entrées et des sorties. Lorsqu'un changement n'est pas validé, la décision prise par l'utilisateur, est formulée par une autre demande de changement toujours au niveau de l'interface d'édition d'ontologies formelles de la plateforme.

L'ontologie de domaine est manipulée au niveau de la plateforme. Le plugin manipule l'ontologie des patrons CMP et l'ontologie journal d'évolution. De plus, son fonctionnement interne nécessite la manipulation de copies volatiles de l'ontologie de domaine (le temps de la gestion d'une transaction de changement) afin d'appliquer temporairement le changement pour l'analyser, vérifier la cohérence des résolutions générées et évaluer leur impact sur la qualité. Pour assurer le passage à l'échelle avec de larges ontologies de domaine, il serait nécessaire de fournir des mécanismes permettant de copier uniquement les parties de concernées par le changement et sa gestion.

[89] http://dafoe4app.fr/

L'interface utilisateur de saisie d'une demande de changement est très importante car les interactions de départ avec l'utilisateur et les champs de saisie prévus dans cette interface, permettront de tenir compte de l'intention du changement et d'assurer une bonne spécification du changement requis surtout s'il est complexe. En effet, les changements complexes ne doivent pas être composés arbitrairement. De même, pour préciser la profondeur des décompositions et assurer une instanciation pertinente par la suite, du patron de changement complexe correspondant, il est nécessaire de prévoir une interaction suffisamment explicite avec l'utilisateur au moment de la demande de changement, sans pour autant le contraindre à se soucier du détail de réalisation du changement, qui reste du rôle essentiel du système.

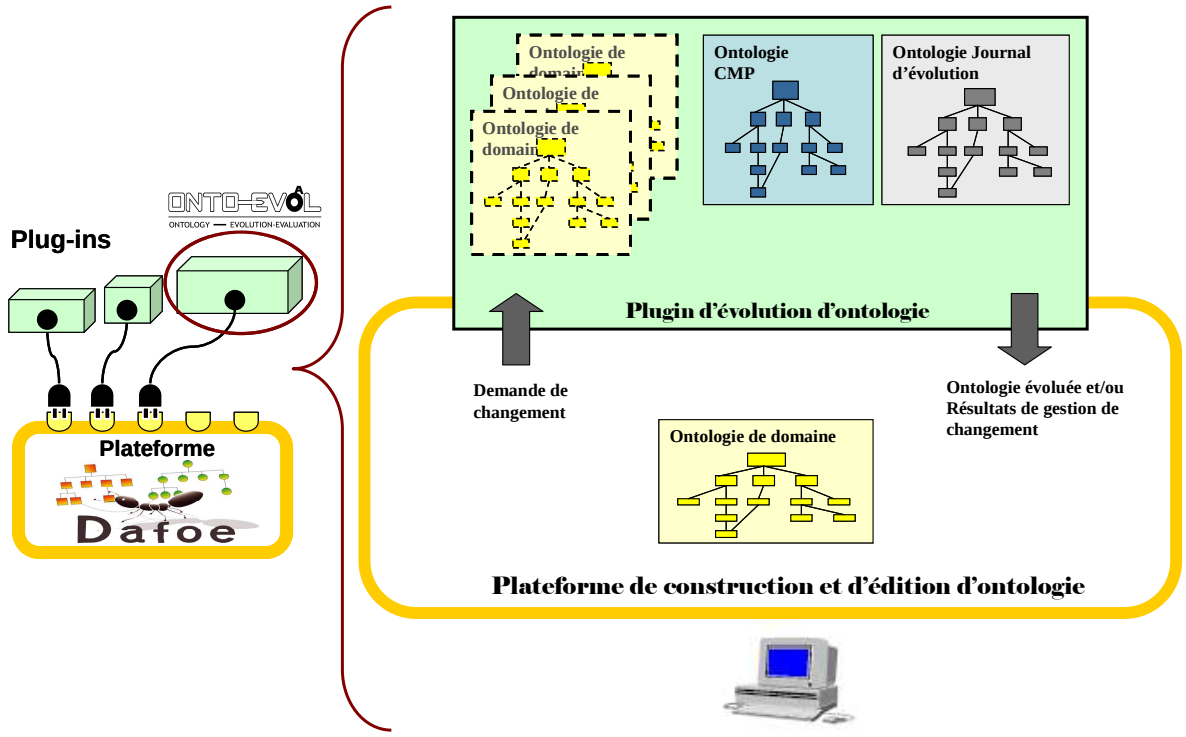


Figure 3. Système d'évolution comme plugin de la plateforme DAFOE.

Assurer une évolution cohérente d'ontologie, en épargnant l'utilisateur des problématiques de gestion de changement liées à son analyse et sa résolution, tout en l'informant des implications sur la cohérence, des résolutions dérivées, et des décisions appliquées pour justifier le changement lui-même, son coût et les résultats de son application (ou non application), sont des besoins fonctionnels essentiels pour le plugin.

La compréhensibilité et la pertinence des interfaces d'affichage des résultats de gestion, sont aussi à considérer, afin d'offrir un feedback clair et cohérent des opérations appliquées, des justifications apportées et des informations les plus importantes à prendre en compte pour comprendre la non validation d'un changement demandé (pourquoi les résolutions générées ne sont pas cohérentes ? pourquoi toutes les résolutions ont un impact négatif sur la qualité ? etc.).

Par ailleurs, la traçabilité des traitements intermédiaires et des évolutions validées doit être assurée afin d'assister l'utilisateur tout en lui permettant de contrôler les résultats du processus. La traçabilité permet également, la réversibilité des modifications appliquées à travers un historique détaillé des opérations appliquées, et garantit l'utilisabilité du système d'évolution.

VII- CONCLUSION

Dans ce chapitre, nous avons présenté l'évaluation de l'approche *Onto-Evo^{a}l*. Elle a été réalisée sur des ontologies de domaine de taille limitée, ce qui a permis d'évaluer les fondements de l'approche sans se soucier des contraintes de passage à l'échelle. Bien que l'évaluation n'ait pas été conduite de manière linéaire sur l'ensemble des phases de l'approche, faute d'une implémentation complète du processus, nous avons néanmoins, tenté de tester l'ensemble des fonctionnalités en combinant des expérimentations manuelles et des expérimentations automatiques employant un prototype implémentant un noyau de l'approche.

De même, nous avons proposé un prototypage pour une architecture complète d'un système d'évolution *Onto-Evo^{a}l* en décrivant les principaux composants logiciels et les types des artefacts produits. Un contexte d'intégration du système à une plateforme de construction et d'édition d'ontologie, comme un plugin d'évolution, a aussi été décrit.

Dans le chapitre suivant, nous synthétisons l'ensemble des travaux développés dans les différents chapitres, et nous proposons un ensemble de perspectives.

Chapitre 8 : Conclusion et perspectives

Les travaux de recherche développés dans cette thèse, définissent une approche d'évolution d'ontologie *Onto-Evo*^{a}*l* (Ontology Evolution-Evaluation) qui s'appuie sur une modélisation de patrons de gestion de changement CMP (Change Management Patterns). Ces patrons spécifient des classes de *changements*, des classes d'*incohérences* et des classes d'*alternatives* de résolution. Sur la base de ces patrons et des liens entre eux, un processus automatisé permettant de conduire l'application des changements tout en maintenant la cohérence de l'ontologie évoluée a été développé. L'approche intègre également une activité d'évaluation basée sur un modèle de qualité d'ontologie. Ce modèle est employé pour guider la gestion des incohérences en évaluant l'impact des résolutions proposées sur la qualité de l'ontologie et ainsi choisir celle qui préserve la qualité de l'ontologie évoluée.

La gestion des changements étant fortement liée au modèle dans lequel est représentée l'ontologie, nous nous sommes focalisés sur le langage OWL en tenant compte de l'impact des changements sur la cohérence logique de l'ontologie telle que spécifiée dans la couche OWL DL.

I- SYNTHESE DES POINTS CLES

Le processus d'évolution *Onto-Evo*^{a}*l* permet : (i) de spécifier formellement une demande de changement, (ii) de diriger et contrôler la gestion des changements, (iii) de maintenir la cohérence de l'ontologie et (iv) de guider les choix de résolution et optimiser l'application des changements en tenant compte de l'impact sur la qualité de l'ontologie.

L'apport de l'approche réside (1) dans la modélisation par patrons guidant le processus d'évolution, (2) l'intégration de l'évaluation de la qualité pour optimiser la résolution des changements et (3) la modélisation formelle et explicite du journal d'évolution.

I.1- Principes de l'approche

L'approche est structurée selon trois niveaux :

- Le niveau *processus* qui représente le cœur de l'approche. Il comprend les composants fonctionnels de l'approche à savoir : les quatre phases principales de gestion de changement et le module d'évaluation de qualité guidant le choix des résolutions ;
- Le niveau *patrons* qui représente le méta-modèle guidant le déroulement du processus. Il est décrit par les patrons *CMP* (patrons de changements, patrons d'incohérences et patrons d'alternatives) et les liens conceptuels entre eux ;
- Le niveau *historique* qui assure la traçabilité des changements et des traitements effectués le long du processus à travers le log d'évolution sauvegardant tout l'historique.

I.2- Patrons de gestion de changement CMP

Les patrons CMP sont introduits comme un moyen de faciliter l'application d'un changement tout en assurant la cohérence de l'ontologie évoluée. Ils modélisent des structures récurrentes de changements, des incohérences logiques qu'ils peuvent potentiellement causer et des alternatives pouvant résoudre ces incohérences constituant ainsi, un potentiel prometteur pour le guidage du processus d'évolution.

Les patrons CMP sont représentés selon deux couches :

- Une couche de présentation sous forme d'un catalogue de patrons CMP décrits en langage naturel et illustrés par des diagrammes UML. Elle facilite l'échange, le partage et la documentation des patrons CMP notamment pour des besoins d'évaluation de ces patrons ;
- Une couche de spécification formelle sous forme d'une ontologie OWL DL des patrons CMP définissant la sémantique des patrons (à travers des subsomptions, des compositions, des restrictions de propriétés), des relations entre eux, et de leur application. Elle facilite le guidage du processus de gestion de changement.

Les patrons CMP guident le processus d'évolution de l'approche *Onto-Evo*^{a}*l* au niveau des trois phases clés : la spécification du changement, l'analyse du changement et la résolution du changement. Ainsi, à la demande d'application d'un changement, l'utilisation des patrons CMP permet de :

- Définir la signification du changement requis (sa spécification explicite, sa portée, etc.) ;
- Le formaliser et l'associer à un patron de changements ce qui constitue sa signature ;
- Déduire – en tenant compte de la structure de l'ontologie et des contraintes du modèle OWL DL – ses impacts attendus (éventuels) sur la cohérence et ce, à partir :
    - du patron de changements instancié : les contraintes à vérifier pour que le changement soit applicable tout en maintenant la cohérence,
    - de la relation conceptuelle entre le patron de changements en question et les patrons d'incohérences qui lui sont liés. Cette relation renseigne sur les incohérences que ce type de changement peut potentiellement causer.
- Détecter ses impacts réels sur la cohérence en se basant sur les schémas d'axiomes de l'ontologie initiale (contraintes logiques exprimées dans l'ontologie). La détection des incohérences réellement causées suite à l'application du changement, fait intervenir un raisonneur. Les résultats d'analyse et les notifications d'incohérences par le raisonneur, sont formatés et complétés pour instancier les incohérences détectées aux patrons d'incohérences correspondants.
- Proposer à partir des patrons d'incohérences instanciés, les patrons d'alternatives pouvant les résoudre en se basant sur la classe *résolution* liant les instances des trois types de patrons et aussi sur la structure de l'ontologie en évolution et les contraintes logiques qui y sont exprimées.

I.3- Résolution des Incohérences logiques

La résolution des incohérences se base sur la localisation des incohérences causées par un changement, leur classification selon les patrons d'incohérences prédéfinis par l'ontologie CMP afin de les expliciter, et la proposition de résolutions en générant, pour chaque incohérence, les alternatives de résolution permettant de la résoudre (patrons d'alternatives).

L'approche *Onto-Evo*^{a}*l* supporte la localisation des incohérences logiques à deux niveaux : la TBox (insatisfiabilité de concept), et la TBox et ABox (aucun modèle dans le domaine ne peut correspondre à l'ontologie), ce qui offre une gestion plus complète des incohérences. La localisation fait appel au raisonneur Pellet mais, est indépendante de son fonctionnement interne (approche black-box).

L'application des patrons CMP en combinaison avec les fonctionnalités offertes par le raisonneur, rendent le guidage du processus de gestion de changements plus efficace.

I.4- Evaluation de la qualité pour guider la résolution de changement

Afin d'aller plus loin dans l'automatisation et l'optimisation, nous avons introduit dans le processus d'évolution, des techniques d'évaluation de qualité que nous avons employées pour évaluer l'impact des solutions de résolution générées par la gestion des changements, sur la qualité de l'ontologie évoluée. L'idée étant de classer les résolutions proposées – quand plusieurs solutions sont possibles – pour n'en choisir que celle qui préserve – si ce n'est améliore – la qualité de l'ontologie et ainsi, guider le choix des résolutions à appliquer plutôt que de solliciter l'intervention de l'ingénieur d'ontologie.

L'évaluation se base sur un modèle de qualité évaluant l'impact des résolutions générées par le processus, sur le contenu et l'usage de l'ontologie à travers un ensemble de métriques quantitatives et ce, afin de choisir une résolution qui préserve la qualité de l'ontologie.

Ainsi, plutôt que de solliciter directement l'expert dans la résolution des incohérences, les résolutions générées sont évaluées par rapport à leur impact sur la qualité de l'ontologie dans le but de guider et d'optimiser la validation des résolutions. La résolution qui préserve la qualité de l'ontologie, peut être automatiquement choisie et les changements directement validés. L'expert ne sera sollicité que si toutes les résolutions ont un impact négatif sur la qualité. Il sera guidé dans sa décision, par l'ensemble des résultats d'analyse, de résolution et d'évaluation.

L'évaluation de la qualité joue un double rôle dans le processus d'évolution :

- Un rôle direct guidant le choix entre plusieurs solutions de résolution d'incohérences ;
- Un rôle indirect puisque l'évaluation de l'impact de l'application – cohérente – du changement sur l'ontologie, en tenant compte de la qualité de l'ontologie initiale (avant évolution), permet d'estimer et justifier le coût du changement.

I.5- Récapitulatif des niveaux de cohérence gérés

Onto-Evo^{a}*l* gère quatre niveaux de cohérence :

- La cohérence structurelle qui se réfère aux contraintes du langage OWL DL et à l'utilisation de ses constructeurs. Elle est assurée à travers la spécification formelle des changements à appliquer par les patrons de changements dont la modélisation s'est basée sur les fondements du modèle OWL DL ;
- La cohérence logique qui tient compte de la sémantique exprimée en DL et des interprétations au niveau instance. Elle est vérifiée et maintenue à travers tout le processus de gestion de changement de *Onto-Evo*^{a}*l* et se matérialise essentiellement par les patrons d'incohérences et les patrons d'alternatives de résolution ainsi que les relations conceptuelles entre eux ;
- La cohérence conceptuelle se référant à la conceptualisation de l'ontologie c'est-à-dire, à la manière dont elle est modélisée mais aussi modifiée/évoluée. Elle correspond à des contraintes de bonne conceptualisation. Ce niveau de cohérence est assuré d'une part, par les patrons de changements et les patrons d'alternatives correspondant à des changements dérivés, et d'autre part, par un ensemble de critères définis dans le modèle de qualité ;
- La cohérence de modélisation de domaine confrontant la conceptualisation de l'ontologie au domaine qu'elle modélise pour voir si elle reflète bien les connaissances du domaine. Cette dimension est prise en compte dans le modèle de qualité à travers un ensemble de critères évaluant l'usage de l'ontologie par rapport aux sources de domaine.

I.6- Traçabilité de l'évolution

Tout au long du processus, les résultats de gestion du changement requis, ainsi que les applications validées sont sauvegardés dans le journal d'évolution. Le journal d'évolution est une structure permettant de conserver l'historique des évolutions de l'ontologie et les détails de traitement de ces évolutions sous forme de séquences chronologiques d'information. Il facilite le contrôle et le suivi de l'évolution, le retour arrière, la justification des changements et la gestion des versions.

Dans le but d'assurer la compatibilité avec le modèle des ontologies de domaine gérées par l'approche *Onto-Evo^{a}l*, et la sémantique de l'ontologie *CMP* guidant le processus et, de disposer d'un format suffisamment explicite et formel pour faciliter ultérieurement l'apprentissage de patrons, nous avons modélisé le journal d'évolution sous forme d'une ontologie OWL DL.

L'ontologie du journal d'évolution modélise les trois dimensions qu'un historique d'évolution devrait prendre en compte à savoir, la dimension *trace d'exécution du processus pour la gestion d'une transaction de changement*, la dimension *trace d'évolution d'une version de l'ontologie* et la dimension *trace de versionning de l'ontologie*.

Le journal d'évolution constitue une véritable base de connaissances, il comprend les entités ontologiques décrivant l'historique d'évolution et les instances correspondantes aux traces d'exécution du processus, d'évolution d'une version de l'ontologie, et celles des différentes versions d'une ontologie.

I.7- Expérimentation et implémentation

L'évaluation de l'approche a été conduite à travers une première évaluation manuelle, appliquée sur un ensemble d'ontologies disponibles sur le Web, et une deuxième évaluation automatique basée sur des expérimentations appliquées en exécutant le prototype implémenté. Le prototype supporte la gestion d'un premier sous ensemble de patrons de changements basiques, et implémente un sous-ensemble des fonctionnalités de l'approche.

Au-delà du prototype, nous avons détaillé l'architecture globale d'un système implémentant l'approche *Onto-Evo^{a}l* en tenant compte des composants logiciels et des artefacts produits.

Les travaux de recherche menés dans le cadre de cette thèse ont été intégrés dans le projet de recherche RNTL DAFOE4App[90] (Differential And Formal Ontology Editor for Applications) développant une plateforme de construction et d'édition d'ontologies. L'objectif du projet est de proposer une méthode complète associée à une plateforme technique pour concevoir des ontologies, de la modélisation à partir du domaine à leur évolution en passant par leur formalisation et exploitation. L'idée est de greffer le système d'évolution d'ontologie comme un plugin à la plateforme DAFOE assurant la gestion d'évolution d'ontologie au niveau de la couche formelle de la plateforme.

II- SYNTHESE PAR RAPPORT AUX TRAVAUX EXISTANTS EN EVOLUTION D'ONTOLOGIE

Après avoir rappelé les fondements de l'approche *Onto-Evo^{a}l* et le fonctionnement des différentes phases de son processus de gestion de changement, nous reprenons dans cette section, le tableau comparatif établi dans le chapitre 2 (section V) pour la synthèse des travaux existants en évolution d'ontologie, afin de positionner l'approche *Onto-Evo^{a}l* par rapport aux caractéristiques considérées lors de l'étude de l'état de l'art.

[90] http://dafoe4app.fr/

Onto-Evo^{a}l **Approche d'évolution d'ontologie guidée par des patrons de gestion de changement**			
Caractéristiques générales	Processus d'évolution		Un processus global de gestion de changement d'ontologie dans un contexte local (de la spécification à l'application de changement en passant par la maintenance de la cohérence).
	Langage d'ontologie		OWL DL.
	Outil / Prototype		Un premier prototype implémentant un sous-ensemble des fonctionnalités de l'approche.
Fonctionnalités	Identification des besoins de changement		
	Spécification des changements		A partir des patrons de changements (changements basiques ou complexes).
	Maintenance de la cohérence	Niveau	Structurelle, logique, conceptuelle, et de modélisation de domaine.
		Vérification	A partir des patrons CMP et du modèle de qualité.
		Proposition de résolution	A partir des patrons d'alternatives de résolution et des relations sémantiques entre les patrons CMP telles que définies dans l'ontologie CMP.
		Résolution automatique	Guidée par l'application des patrons CMP et par l'évaluation de l'impact des résolutions sur la qualité de l'ontologie évoluée.
	Propagation des changements	Cible	
		Type	
	Détection des changements		
	Versioning	Version évoluée	Génération d'une version évoluée.
		Comparaison des versions	
		Gestion de plusieurs versions	Possible à partir de la trace des différentes versions d'une ontologie enregistrée dans le journal d'évolution.
	Log d'évolution	Changements appliqués	Oui (niveau entités et niveau ontologie).
		Trace des opérations de gestion	Traçabilité de tout le processus de gestion de changement.
	Spécificités		Processus guidé par l'application des patrons de gestion de changements CMP et par l'évaluation de l'impact des résolutions sur la qualité de l'ontologie évoluée.

Table 1. Synthèse de l'approche *Onto-Evo^{a}l*.

III- PERSPECTIVES

La synthèse des travaux développés dans cette thèse et des contributions apportées par l'approche *Onto-Evo[a]l*, ouvre de nombreuses perspectives de dissémination, d'amélioration et d'extension. Ainsi, dans les sections suivantes, nous présentons une initiative d'intégration des patrons CMP au portail des patrons de conception d'ontologie ODP[91], nous discutons des améliorations possibles pour assurer le passage à l'échelle de l'approche, puis nous présentons des propositions pour étendre les patrons CMP à d'autres modèles d'ontologie et aussi, les enrichir par l'apprentissage de nouveaux patrons.

III.1- CMP et ODP

Une des premières perspectives à court terme est d'intégrer les patrons CMP au portail des Ontology Design Patterns ODP. Ceci permettra de mettre à disposition une librairie de patrons de gestion de changement pour les partager, les évaluer et les certifier à travers le portail. Les patrons CMP peuvent être intégrés comme un type particulier de patrons ODP et mis en relation avec certains types de patrons ODP. Ainsi, en rappelant qu'un *Content Pattern* décrit comment modéliser une sous-ontologie par rapport à un domaine donné, un patron de changement peut correspondre à un *Content Pattern* pour le domaine « ontologie ». De même, un *Content Pattern* peut être décrit par un patron de changement complexe composé de plusieurs changements basiques et/ou complexes. L'avantage de cette mise en relation est double, le patron de changement décrit comment appliquer l'ajout d'un *Content Pattern* à une ontologie existante (la composition de changement réalisant un *Content Pattern*). De plus, à travers les relations sémantiques avec les patrons d'incohérences et d'alternatives (si on considère l'ensemble complet des patrons CMP), la cohérence de l'ontologie à laquelle est appliqué le *Content Pattern* peut être maintenue. L'objectif premier d'ailleurs, de la modélisation des CMP est bien de guider le processus de gestion de changement.

Un deuxième type de relation entre les patrons CMP et les patrons ODP est défini entre les patrons d'alternatives de CMP et les patrons logiques *Logical Patterns* de ODP. Les patrons *Logical Patterns* proposent des solutions à des problèmes de conception d'ontologie que les primitives ou les constructeurs du langage de représentation ne supportent pas. Selon cette optique, les patrons d'alternatives sont proposés comme un type particulier de *Logical Patterns* permettant de résoudre des problèmes d'incohérences logiques dans la conception d'ontologies.

Le patron d'alternatives « *définir une classe hybride pour résoudre une disjonction de subsomption* » détaillé dans la section IV.3.1 du chapitre 4, et proposé comme patron *Logical Pattern* sur le portail de patrons ODP[92], a d'ailleurs été accepté dans le cadre du workshop WOP2009[93] (Workshop on Ontology Patterns), et fait l'objet d'une publication.

III.2- Passage à l'échelle de l'approche

Outre la perspective d'enrichir et compléter le prototype développé pour en faire un système d'évolution implémentant l'ensemble des fonctionnalités de l'approche *Onto-Evo[a]l*, le passage à l'échelle de l'approche sur le plan logiciel et aussi, au niveau de ses méta-modèles (patrons) est à considérer.

L'approche *Onto-Evo[a]l* implique des activités lourdes en termes de coût de traitement et de capacité mémoire (application temporaire de changements, classification et mise en correspondance de patrons, instanciation de patrons extraits de l'ontologie CMP, processus récursif de résolution d'incohérences,

[91] http://ontologydesignpatterns.org
[92] http://ontologydesignpatterns.org/wiki/Submissions:Define_Hybrid_Class_Resolving_Disjointness_due_to_Subsumption
[93] http://ontologydesignpatterns.org/wiki/WOP2009:Main

évaluation de l'impact des résolutions sur la qualité, sauvegarde de tous les résultats de traitement dans le journal d'évolution, etc.), les choix d'implémentation doivent tenir compte des contraintes d'applicabilité (temps de réponse, capacité mémoire nécessaire, format de stockage, etc.) de ces activités afin de supporter la gestion de changements complexes et surtout, le passage à l'échelle pour gérer l'évolution de larges ontologies.

Bien qu'elle ne soit pas prise en compte dans la version actuelle de l'approche *Onto-Evo*^{a}*l*, la dimension « *modularité de l'ontologie* » est très importante aussi bien en tant que critère d'évaluation de qualité (chapitre 6, section V.4) qu'au niveau de la gestion même du changement. Il serait beaucoup plus pratique de gérer un changement sur un module d'une ontologie que sur l'ontologie dans sa globalité particulièrement lorsqu'il s'agit de larges ontologies. Ceci permet de mieux appréhender l'impact du changement et le résoudre.

L'idée serait qu'à la spécification du changement, nous définissions des mécanismes qui permettent d'extraire de l'ontologie le module concerné par le changement. Ces mécanismes pourraient s'inspirer des travaux sur le partionnement d'ontologies à aligner pour faciliter l'alignement d'ontologies de grande taille (Hamdi et al., 2009). Le principe est de limiter la taille des ensembles de concepts en entrée de l'outil d'alignement en partionnant les deux ontologies à aligner en plusieurs blocs. Cependant, l'identification de la partie de l'ontologie concernée par le changement et sa gestion doit tenir compte non seulement des relations de subsomption entre les classes, mais aussi des propriétés, des instances et des axiomes.

III.3- Dépendance à OWL et méta-modélisation générique des CMP

La modélisation des hiérarchies de patrons de changements, d'incohérences et d'alternatives est spécifique au modèle OWL DL. Néanmoins, certaines propriétés de description et certaines classes abstraites assez générales de l'ontologie CMP, peuvent être adaptées et réutilisées pour la gestion de changement d'autres modèles ontologiques que OWL.

Il serait intéressant de définir une couche générique dans la modélisation des patrons CMP qui soit indépendante du langage, sous laquelle, une couche spécifique plus formelle exprimée en OWL DL, permet l'instanciation et le codage des patrons CMP.

Une autre perspective tout aussi intéressante serait d'adapter les patrons CMP pour prendre en compte l'expressivité de OWL 2[94] surtout que les extensions proposées (Grau et al, 2008) sont aujourd'hui supportées par des raisonneurs tels que Pellet, l'API OWL API et des éditeurs tels que Protégé 4.0.

III.4- Apprentissage de patrons CMP

Les patrons CMP définis couvrent les changements OWL DL basiques et un sous-ensemble de changements complexes, un premier noyau d'incohérences logiques et des patrons d'alternatives permettant de les résoudre. Les patrons de changements basiques sont définis directement à partir du modèle OWL DL et enrichis par les contraintes qui doivent être satisfaites pour les appliquer. Les changements complexes par contre, sont infinis. Il y a toujours une nouvelle possibilité de combinaison de changements spécifiant un nouveau changement complexe.

Les patrons d'incohérence définis correspondent à des incohérences logiques élémentaires. Ils peuvent certainement être combinés pour modéliser des incohérences plus complexes. Notre objectif, dans le cadre de cette thèse, était d'expliciter les principaux types d'incohérences logiques pouvant être causées pour guider le processus de gestion de changement et particulièrement la résolution des incohérences. Fournir des modélisations formelles pour des incohérences logiques est un premier pas dans la gestion

[94] http://www.w3.org/TR/owl2-syntax/

automatisée des effets de changements. En perspective future, la modélisation des patrons d'incohérences – de même que les autres types de patrons CMP – pourrait être enrichie par de nouveaux types d'incohérence élémentaires et aussi par des compositions d'incohérences, notamment à partir du module d'apprentissage prévu dans l'architecture de l'approche *Onto-Evo*^{a}*l*.

Le rôle du module d'apprentissage est de compléter et enrichir les patrons CMP à travers l'application de l'approche sur des cas d'évolution réels. En effet, il n'est pas évident de fournir un système de gestion complet qui gère tous les types de changements et spécifie d'avance toutes les solutions de résolution. L'intention de notre approche est de fournir un système capable de conduire de manière automatisée la gestion des changements en maintenant la cohérence et qu'il soit lui-même évolutif à travers un module d'apprentissage enrichissant ses propres modèles (patrons).

Le module d'apprentissage de patrons exploiterait ainsi le journal d'évolution qui constitue une véritable base de connaissance d'évolution, pour enrichir les patrons CMP existants et apprendre de nouveaux patrons et de nouvelles relations de résolutions.

Liste des publications

Chapitre de livre

- R. Djedidi, M.-A. Aufaure, « Ontology Evolution: State of the Art and Future Directions », In Book *Ontology Theory, Management and Design: Advanced Tools and Models*, F. Gargouri et W. Jaziri (Eds.), IGI Global Publication. (A paraître en 2010).

Publications en Conférences Internationales

- R. Djedidi, M.-A. Aufaure (2010) « Onto-Evo[a]l an Ontology Evolution Approach Guided by Pattern Modelling and Quality Evaluation», Proceedings of the Sixth International Symposium on Foundations of Information and Knowledge Systems (FoIKS 2010), February 15-19 2010, Sofia, Bulgaria (Accepté)
- R. Djedidi, M-A. Aufaure, « Ontology Change Management », In A. Paschke, H. Weigand, W. Behrendt, K. Tochtermann, T. Pellegrini (Eds.), the 5th International Conference on Semantic Systems (I-Semantics 09), Proceedings of I-KNOW '09 and I-SEMANTICS '09, ISBN 978-3-85125-060-2, pp 611-621, Verlag der Technischen Universitt Graz. Graz, Austria, 2 - 4 September 2009.
- R. Djedidi, M-A. Aufaure, « Ontological Knowledge Maintenance Methodology», Proceedings of the 12th International Conference Knowledge-Based Intelligent Information and Engineering Systems (KES 2008), Ignac Lovrek, Robert J. Howlett, and Lakhmi C. Jain (Eds.), Part I. Lecture Notes in Computer Science 5177 Springer 2008, ISBN 978-3-540-85562-0, p. 557-564, Zagreb, Croatie, 3-5 Septembre, 2008.
- R. Djedidi, M.-A. Aufaure, « Medical Domain Ontology Construction : a Basis for Medical Decision Support », Proceedings of the 20th IEEE International Symposium on Computer-Based Medical Systems (CBMS'07), Special track on Healthcare Knowledge Management, p. 509-511, Maribor, Slovénie, 20-22 juin 2007.

Publications en Workshops Internationaux

- R. Djedidi, M.-A. Aufaure, « Change Management Patterns (CMP) for Ontology Evolution Process », Proceedings of the 3rd International Workshop on Ontology Dynamics (IWOD2009) in ISWC 2009, M. D'Aquin & G. Antoniou (Eds.), CEUR Workshop Proceedings, Vol. 519, Washington DC, USA, 26 Octobre, 2009. http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-519/
- R. Djedidi, M.-A. Aufaure, « Define Hybrid Class Resolving Disjointness Due to Subsumption - http://ontologydesignpatterns.org/wiki/Submissions:Define_Hybrid_Class_Resolving_

  Disjointness_due_to_Subsumption», Proceedings of the 1st Workshop on Ontology Patterns (WOP 2009) in ISWC 2009, E. Blomqvist, K. Sandkuhl, F. Scharffe, V. Svatek (Eds.), CEUR Workshop Proceedings, Vol. 516, pp:100-103, Washington DC, USA, 25 Octobre, 2009. http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-516/
- R. Djedidi, M.-A. Aufaure, « OWL Change Management Patterns »,

Proceedings of the 5th Workshop on Semantic Web Applications and Perspectives (SWAP2008), CEUR Workshop Proceedings, A. Gangemi, J. Keizer, V. Presutti, and H. Stoermer (ed.), ISSN 1613-0073 LNCS/LNAI Springer-Verlag, Rome, Italy, December 15-17, 2008.

Online http://ceur-ws.org/Vol-426/swap2008_submission_41.pdf.

Publications en Conférences Francophones

- R. Djedidi, M.-A. Aufaure, «Patrons de gestion de changements OWL», 20èmes journées francophones d'Ingénierie des Connaissances (IC'09), Actes d'IC, Fabien Gandon (Ed.), ISBN 978-2-7061-1538-7, PUG, p. 145-156, Hammamet, Tunisie, 25-29 mai 2009.
- R. Djedidi, M.-A. Aufaure, «Enrichissement d'ontologies : maintenance de la consistance et évaluation de la qualité », 19èmes journées francophones d'Ingénierie des Connaissances (IC'08), Nancy, France, 18-20 juin 2008.
- R. Djedidi, H. Abboute, M.-A Aufaure, « Evolution d'ontologies : Validation des changements basée sur l'évaluation », Proceedings des 1ères journées francophones sur les ontologies "Les ontologies : mythes, réalités et perspectives" (JFO 2007), édition du centre de Publication Universitaire, F. Gargouri, D. Benslimane et P. Bourque, Sousse, Tunisie, 18-20 octobre 2007.
- R. Djedidi H., S. Ben Lagha, M. Ben Ahmed, « Utilisation des technologies XML pour la formalisation de l'ontologie de modèles e-business », Proceedings des 5èmes journées francophones "Extraction et Gestion des Connaissances" (EGC'05), Publication format Poster, Editions Cépaduès, Revue des Nouvelles Technologies de l'Information RNTI-E-3, Paris, France, 19-21 janvier 2005.

Publications en Workshops Francophones

- R. Djedidi, M.-A. Aufaure, « Gestion des changements d'une ontologie : Résolution d'inconsistances guidée par l'évaluation de la qualité », actes de l'atelier Modélisation des Connaissances (ModCo'08), 8èmes journées francophones "Extraction et Gestion des Connaissances" (EGC'08), INRIA Sophia Antipolis, France, 29 janvier-1er février 2007.
- R. Djedidi, M.-A. Aufaure, « Une approche générique de construction d'ontologies de domaine à partir de sources hétérogènes », actes de l'atelier Modélisation des Connaissances (ModCo'07), 7èmes journées francophones "Extraction et Gestion des Connaissances" (EGC'07), Namur, Belgique, 23-26 janvier 2007.
- R. Djedidi, M.-A. Aufaure, « Une approche de construction d'ontologies décisionnelles pour une aide à la décision médicale », 1er Atelier Systèmes Décisionnels (ASD'06), édition sur CD, Agadir, Maroc, 8 décembre 2006.

BIBLIOGRAPHIE

Alani, H., Brewster, C., Shadbolt, N. Ranking Ontologies with AKTivRank. (2006) Proceedings of the 5th International Semantic Web Conference ISWC'05. LNCS Vol. 4273, pp: 1-15. Springer. http://eprints.aktors.org/556/01/iswc06-camera-ready.pdf

Alchourron, C., Gärdenfors, P., & Makinson, D. (1985). On the logic of theory change: partial meet contraction and revision functions. *Journal of Symbolic Logic*, 50(2), 510-530.
URL: http://www.jstor.org/stable/2274239

Aranguren, M.E., Antezana, E., Kuiper, M., Stevens, R. (2007). Ontology Design Patterns for bio-ontologies: a case study on the Cell Cycle Ontology. Proceedings of the 10th Bio-Ontologies Special Interest Group Workshop, Vienna, Austria. Published April 2008, BMC Bioinformatics, 9(Suppl 5):S1. doi:10.1186/1471-2105-9-S5-S1. http://www.biomedcentral.com/1471-2105/9/S5/S1

Arndt, R., Troncy, R., Staab, S., Hardman, L, Vacura, M. (2007). Designing a well-founded multimedia ontology for the web. In Proceedings of the 4th European Semantic Web Conference (ISCW'07), Busan Korea, November. Springer.

Auer, S., Herre, H. (2007) A Versioning and Evolution Framework for RDF Knowledge Bases. In: A. Voronkov, I. Virbitskaite (Eds.) Perspectives of Systems Informatics. Revised Papers of the 6th International Andrei Ershov Memorial Conference. LNCS: Vol. 4378. PSI, pp. 55–69. Springer-Verlag, Berlin. Lien : http://www.informatik.uni-leipzig.de/~auer/publication/PSI-evolution.pdf.

Aussenac-Gilles, N., Jacques, M.P. (2006) Designing and Evaluating Patterns for Ontology Enrichment from Texts. EKAW 2006: 158-165.

Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., Patel-Schneider, P. F. (Eds.) (2003). The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, New York, USA. DOI: 10.2277/0521781760

Baader, F., Suntisrivaraporn, B. (2008). Debugging SNOMED CT using axiom pinpointing in the description logic $\mathcal{EL}+$. In Proceedings of KR-MED'08: *Representing and Sharing Knowledge Using SNOMED*.

Banerjee, J., Kim, W., Kim, H.J., & Korth, H. (1987). Semantics and implementation of schema evolution in object-oriented databases. ACM SIGMOD Record: Vol. 16(3), (pp. 311-322), New York, USA: ACM. doi: http://doi.acm.org/10.1145/38714.38748

Baneyx, A., Charlet, J., Jaulent, M-C. (2005). Construction d'ontologies médicales fondée sur l'extraction terminologique à partir de textes : application au domaine de la pneumologie. Journées Francophones d'Informatique Médicale. JFIM 2005.

Bloehdorn, S., Haase, P., Sure, Y. & Voelker, J. (2006). Ontology evolution. In J. Davies, R. Studer, & P. Warren (Ed.s), *Semantic Web Technologies, Trends and research in Ontology-based Systems* (pp. 51-70). John Wiley & Sons Publication. doi: 10.1002/047003033X.ch4

Blomqvist, E. (2005). Fully automatic construction of enterprise ontologies using design patterns: Initial method and first experiences. In R. Meersman, Z. Tari, M.-S. Hacid, J. Mylopoulos, B. Pernici, zalp

Babaoglu, H.-A. Jacobsen, J. P. Loyall, M. Kifer, and S. Spaccapietra, Eds. OTM Conferences (2), LNCS: Vol. 3761, pages 1314–1329. Springer.

Blomqvist, E. (2009) Semi-automatic Ontology Construction based on Patterns. PhD Thesis. Linköping University, Department of Computer and Information Science.
http://liu.diva-portal.org/smash/record.jsf?searchId=1&pid=diva2:207543

Blundell, B., & Pettifer, S. (2004). Graph visualization to aid ontology evolution in Protégé. *Proceedings of the 7th International Protégé Conference*. Bethesda, Maryland.
Retrieved from http://protege.stanford.edu/conference/2004/posters/Blundell.pdf

Borst, W.N. (1997). Construction of Engineering Ontologies for Knowledge Sharing and Reuse. Thèse. Centre for Telematica and Information Technology, University of Tweenty. Enschede, The Netherlands. http://doc.utwente.nl/17864/1/t0000004.pdf

Brachman, Schmolze (1985). An overview of the KL-ONE knowledge representation system. *Cognitive Science*, V(9), pp.171-216.

Brank J., Grobelnik M., Mladenic D. (2005). A Survey of Ontology Evaluation Techniques, Proceedings of the Conference on Data Mining and Data Warehouses (SiKDD 2005).

Breche, P., & Wörner, M. (1995). How to remove a class in an object database system. *Proceedings of the 2nd international conference on applications of databases (ADB-95)*(pp. 235-248). Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.45.40&rep=rep1&type=url&i=0
doi : 10.1.1.45.40

Brewster, C., Alani, H., Dasmahapatra, S., Wilks, Y. (2004) Data Driven Ontology Evaluation, in Proceedings of the Lexical Resources and Evaluation Conference (LREC'04), pp. 641–644.

Brockmans, S., Volz, R., Eberhart, A., Löffler, P. (2004). Visual Modeling of OWL DL Ontologies Using UML. In S.A. McIlraith et al. (Eds.), LNCS: Vol. 3298. ISWC 2004, pp. 198–213. Springer-Verlag, Berlin. Lien : http://www.aifb.uni-karlsruhe.de/WBS/sbr/publications/iswc04%20sbr.pdf

Burton-Jones A., Storey V.C., Sugumaram V., Ahluwalia P. (2004) A semiotic metrics suite for assessing the quality of ontologies, Data and Knowledge Engineering.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M. (1996). Pattern-oriented Software Architecture - A System of Patterns. John Wiley & Sons, Chichester.

Castano, S. (2006). Ontology evolution: The BOEMIE approach. *BOEMIE Workshop, at the conference EKAW 06.*
Retrieved from http://www.boemie.org/sites/default/files/pres_castano_boemie.pdf

Castano, S., Espinosa, S., Ferrara, A., Karkaletsis, V., Kaya, A., Melzer, S., ... (2007). Ontology dynamics with multimedia information: The BOEMIE evolution methodology. In G. Flouris, & M. d'Aquin (Eds.) *Proceedings of the International Workshop on Ontology Dynamics (IWOD-07)* at *ESWC 07 Conference* (pp. 41-54). Retrieved from http://kmi.open.ac.uk/events/iwod/iwod-proceedings.pdf

Chagnoux, M., Hernandez, N., Aussenac-Gilles, N. (2008). An interactive pattern based approach for extracting non-taxonomic relations from texts. In P. Buitelaar, P. Cimiano, G. Paliouras and M. Spilliopoulou Eds. Proceedings of the workshop on Ontology Learning and Population associated to

ECAI 2008 et OLP 2008. Patras (Greece). p. 1-6. ftp://ftp.irit.fr/IRIT/IC3/OLP2008_ChagnouxHernandezAussenac.pdf

Charlet, J., Bachimont, B., Troncy, R. (2004). Ontologies pour le Web sémantique. Revue 3I : Information – Interaction – Intelligence, Hors Série 2004. http://www.revue-i3.org/hors_serie/annee2004/revue_i3_hs2004_01_04.pdf

Charlet, J., Szulman, S., Aussenac-Gilles, N., Nazarenko, A., Hernandez, N., Nada, N., et al. (2009). Apport des outils de TAL à la construction d'ontologies : propositions au sein de la plateforme DAFOE (démonstration). Dans : Traitement Automatique des Langues Naturelles (TALN 2009), Senlys (France), Vol. tome 2, A. Nazarenko, T. Poibeau (Eds.), LIPN - Université Paris 13, p. 461-463. Accès : http://www-lipn.univ-paris13.fr/taln09/pdf/TALN_164.pdf

Charlet, J., Szulman, S., Pierra, G., Nadah, N., Teguiak, H. V., Aussenac-Gilles, N., et al. (2008). DAFOE: a multimodel and multimethod platform for building domain ontologies. in D. Benslimane, (Ed.), 2èmes Journées Francophones sur les Ontologies, JFO'08, ACM.

Chen, C., Matthews, M.M. (2008) Metrics for Evaluating the Semantic Implications of Changes in Evolving Ontologies. In Proceedings of The 2008 International Conference on Semantic Web and Web Services, p.76-82. http://www.cis.udel.edu/~chenc/pubs/SWW4275.pdf

Cimiano, P. (2007). On the relation between ontology learning, engineering, evolution and expressivity. *Invited talk at 7th Meeting on Terminology and Artificial Intelligence TIA 2007*. Sophia Antipolis, France. Retrieved from http://www.aifb.uni-karlsruhe.de/WBS/pci/home/Publications/2007/tia07/tia07_slides.pdf

Cimiano, P., & Völker, J. (2005). Text2Onto - a framework for ontology learning and data-driven change Discovery. In A. Montoyo, R. Munoz, & E. Metais (Eds.), LNCS: Vol. 3513. *Natural Language Processing and Information Systems* (pp. 227-238). Berlin, Germany: Springer. doi: 10.1007/b136569

Clark, P., Thompson, J., Porter, B. (2000). Knowledge Patterns. In A. G. Cohn, F. Giunchiglia, and B. Selman Eds. KR2000: Principles of Knowledge Representation and Reasoning, pages 591–600, San Francisco, Morgan Kaufman.

Coenen, F., & Bench-Capon, T. (1993). Maintenance of knowledge-based systems. *Academic Press, the A.P.I.C. Series*, Number 40, ISBN: 0-12-178120-8.

Corby, O., Dieng-Kuntz, R. et Faron-Zucker, C. (2004).Querying the Semantic Web with the CORESE search engine. In R. Lopez de Mantaras and L. Saitta eds, Proceedings. of the 16th European Conference on Artificial Intelligence (ECAI'2004), IOS Press, p.705-709.

Corcho, O, Roussey, C., Vilches Blazquez, L.M. (2009). Catalogue of anti-patterns for formal ontology debugging. Atelier construction d'ontologies - Guide de bonnes pratiques GBPOnto, dans le cadre de la conférence francophone ingénierie de connaissances IC'09.

d'Aquin, M., Sabou, M., Motta, E. (2006). Modularization: a Key for the Dynamic Selection of Relevant Knowledge Components. In Proceeding of the ISWC 2006 Workshop on Modular Ontologies.

d'Aquin, M., Haase, P., Rudolph, S., Euzenat, J., Zimmermann, A., Dzbor, M., et al. (2008). NeOn Formalisms for Modularization: Syntax, Semantics, Algebra. Délivrable D1.1.3 du projet NeOn EU-IST-2005-027595.

Daga, E., Presutti, V., Salvati, A. (2008). Http://ontologydesignpatterns.org [ODP] and evaluation wikiflow. In CEUR Workshop Proceedings of the 5th Workshop on Semantic Web Applications and Perspectives (SWAP2008), Rome, Italy, December 15-17, ISSN 1613-0073, http://ceur-ws.org/Vol-426/swap2008_submission_63.pdf.

Damjanovic, V., (2009). Reengineering Patterns for Ontologizing the Business Processes, In Proceedings of the I-SEMANTICS'09 Conference, Graz, Austria, September 02-04

Davies, J., Fensel, D., Van Harlemen, F. (2003). Towards the semantic Web: Ontology-Driven Knowledge Management. John Wiley & Sons. ISBN-10: 0470848677.

Dividino, R., & Sonntag, D. (2008). Controlled ontology evolution through semiotic-based ontology evaluation. *Proceedings of the 2nd Workshop on Ontology Dynamics, (IWOD-08) at ISWC 08 Conference* (pp. 1-14). Retrieved from http://www.ics.forth.gr/~fgeo/Publications/IWOD-08_Proceedings.pdf

Djedidi, R., (2005). Utilisation des technologies XML pour la conception et la mise en œuvre d'un Handbook de modèles e-business. Master de recherche, Ecole Nationale des Sciences de l'Informatique ENSI, Université de La Manouba.

Drummond, N., Rector, A., Stevens, R., Moulton, G., Horridge1, M., Wang, H. H., Seidenberg, J. (2006). Putting OWL in Order: Patterns for Sequences in OWL. 2nd OWL Experiences and Directions Workshop OWLED'06, Athens, GA. http://www.webont.org/owled/2006/acceptedLong/submission_12.pdf

Duineveld, A.J., Stoter, R., Weiden, M.R. , Kenepa, B., & Benjamins, V.R. (2000). WonderTools? A comparative study of ontological engineering tools. *International Journal of Human-Computer Studies*, 52(6), 1111-1133.

Eder, J., & Wiggisser, K. (2007). Change detection in ontologies using DAG comparison. In J.Krogstie, A.L. Opdahl, & G. Sindre, (Eds.), LNCS: Vol. 4495. *Advanced Information Systems Engineering* (pp. 21-35). Berlin, Germany: Springer. doi: 10.1007/978-3-540-72988-4

Fernandez, E. B., Yuan, X. (2000). Semantic Analysis Patterns. In Proceedings of the 19th International conference on conceptual Modelling, ER2000, pages 183–195, Salt Lake City.

Fernandez-Lopez, M., Gomez-Perez, A., Juristo, N. (1997). METHONTOLOGY: from ontological art towards ontological engineering. In Proceedings of the Spring Symposium Series on Ontological Engineering (AAAI'97), pp 33-40.

Fernandez-Lopez, M., Gomez-Perez, A., Sierra, (1999). Building a chemical ontology using Methontology and the Ontology design environment, *IEEE Intelligent Systems*, 14(1).

Flouris, G. (2006). On belief change and ontology evolution. Ph.D. Thesis, University of Crete, Department of Computer Science, Heraklion, Greece.

Flouris, G., & Plexousakis, D. (2006). Bridging ontology evolution and belief change. LNCS: Vol. 3955, *Advances in Artificial Intelligence* (pp. 486-489). Berlin, Germany: Springer.
doi: 10.1007/11752912_51.

Flouris, G., Huang, Z., Pan, J.Z., Plexousakis, D., Wache, H. (2006b). Inconsistencies, negations and changes in ontologies. In Proceedings of AAAI'06.

Flouris, G., Plexousakis, D., & Antoniou, G. (2005). On applying the AGM theory to DLs and OWL. In Y.Gil, E. Motta, V. Benjamins, & M. Musen, (Eds.), LNCS: Vol. 3729. *The Semantic Web – ISWC 2005* (pp. 216-231). Berlin, Germany: Springer. doi: 10.1007/11574620

Flouris, G., Plexousakis, D., & Antoniou, G. (2006a). Evolving ontology evolution. *Invited Talk at the 32nd International Conference on Current Trends in Theory and Practice of Computer Science* (SOFSEM-06). Retrieved from http://www.sofsem.cz/sofsem06/data/prezentace/23/A/dimitris.pdf

Foo, N. (1995). Ontology revision. In G. Ellis, R. Levinson, W. Rich, & J. F. Sowa (Eds.), *Proceedings of The 3rd International Conference on Conceptual Structures* (pp. 16-31). London, UK: Springer-Verlag. ISBN:3-540-60161-9

Fowler, M. (2003). Patterns of Enterprise Application Architecture. With contributions from Rice, D., Foemmel, M., Hieatt, E., Mee, R. and Stafford, R. Addison-Wesley.

Fox, M.S., Barbuceanu M., Gruninger M., Lin J. (1998) An organization ontology for enterprise modelling, *Simulating organizations*, MIT Press.

Franconi, E., Grandi, F., & Mandreoli, F. (2000). A semantic approach for schema evolution and versioning in object-oriented databases. LNCS: Vol. 1861. *Computational Logic — CL 2000* (pp. 1048-1062). Berlin, Germany: Springer. doi: 10.1007/3-540-44957-4

Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1995). Design Patterns – Elements of Reusable Object-Oriented Software. Addison-Wesley.

Gangemi A., Catenacci C. & Battaglia M. (2004). Inflammation ontology design pattern: an exercise in building a core biomedical ontology with descriptions and situations. In D.M. Pisanelli (Ed.) Ontologies in Medicine. IOS Press, Amsterdam.

Gangemi A., Catenacci C., Ciaramita M., Gil R., Lehmann J. (2005) Ontology evaluation: A review of methods and an integrated model for the quality diagnostic task.

Gangemi A., Gomez-Perez A., Presutti V. & Suarez-Figueroa, M.C. (2007). Towards a Catalog of OWL-based Ontology Design Patterns, CAEPIA 07, Publications du projet Neon (http://www.neon-project.org).

Gangemi, A. (2005). Ontology Design Patterns for Semantic Web Content. In M. Musen et al. (eds.): Proceedings of the 4th International Semantic Web Conference, Galway, Ireland, Springer.

Gangemi, A. (2007). Design Patterns for Legal Ontology Construction. In P. Casanovas, P. Noriega, D. Bourcier, F. Galindo (Eds.), Trends in Legal Knowledge: The Semantic Web and the Regulation of Electronic Social Systems, European Press Academic Publishing. http://ceur-ws.org/Vol-321/paper4.pdf

Gangemi, A., Catenaccia, C., Ciaramita, M., Lehmann, J. (2006a). Modelling ontology evaluation and validation. In Y. Sure and J. Domingue, Eds., Proceedings of the 3rd European Semantic Web Conference (ESWC2006), LNCS Vol. 4011, Springer-Verlag.

Gangemi, A., Catenaccia, C., Ciaramita, M., Lehmann, J. (2006b). Qood grid: A meta-ontology-based framework for ontology evaluation and selection. Proceedings of Evaluation of Ontologies for the Web, 4th International EON Workshop, located at WWW2006 http://km.aifb.uni-karlsruhe.de/ws/eon2006/eon2006gangemietal.pdf

Gangemi, A., Presutti, V. (2007). Ontology design for interaction in a reasonable enterprise. In P. Rittgen, Eds., Handbook of Ontologies for Business Interaction. IGI Global, Hershey, PA, November.

Garcia-Silvia, A., Gomez-Perez, A., Suarez-Figueroa, M.C., Villazon-Terrazas, B. (2008). A Pattern Based Approach for Re-engineering Non-Ontological Resources into Ontologies. In J. Domingue and C. Anutariya (Eds.), ASWC 2008, LNCS 5367, pp. 167–181, Springer-Verlag Berlin Heidelberg.

Gomez-Perez, A. (2004) Ontology Evaluation, In S. Staab and R. Studer, editors. Handbook on Ontologies. International Handbooks on Information Systems. chapitre 13, pp 251-274. Springer-Verlag. ISBN : 3-540-40834-7

Gomez-Perez, A., Fernandez-Lopez, M., Corcho, O. (2004). Ontological Engineering. Springer. ISBN: 1-85233-551-3.

Gomez-Romero, J., Bobillo, F., Delgado, M. (2007). An ontology design pattern for representing relevance in owl. In K. Aberer, K.-S. Choi, and N. Noy, editors, The 6th International Semantic Web Conference and the 2nd Asian Semantic Web Conference 2007, Busan, Korea, November.

Grau, B. C., Horrocks, I., Motik M., Parsia B., Patel-Schneider, P., Sattler, U. (2008). OWL2: The Next Step for OWL. Web Semantics: Science, Services and Agents on the World Wide Web. Vol. 6 , Issue 4. pp: 309-322. Elsevier Science Publishers.
http://www.comlab.ox.ac.uk/people/ian.horrocks/Publications/download/2008/CHMP+08.pdf

Grüber, T.R. (1993). A translation approach to portable ontology specification. Knowledge acquisition 5(2):199-220. http://tomgruber.org/writing/ontolingua-kaj-1993.pdf

Grüninger, M., Fox, M.S. (1995). Methodology for the design and evaluation of ontologies. In Proceedings of the Workshop on Basic Ontological Issues on Knowledge Sharing at the International Joint Conference on Artificial Intelligence (IJCAI'1995).

Guarino N., & Welty C. (2002). Evaluating ontological decisions with OntoClean. *In Communication of the ACM (CACM)*, 45(2), pp: 61-65. New York, US: ACM. Retrieved from http://doi.acm.org/10.1145/503124.503150

Guarino, N., Welty, C. (2004) An Overview of OntoClean. In S. Staab and R. Studer Eds. *Handbook on Ontologies*. International Handbooks on Information Systems. chapitre 8, pp 151-172. Springer-Verlag. ISBN : 3-540-40834-7. at http://www.loa-cnr.it/Papers/GuarinoWeltyOntoCleanv3.pdf

Guizzardi, G. (2005). Ontological foundations for structural conceptual models. PhD thesis, University of Twente, Enschede, The Netherlands, Enschede, October.

Guizzardi, G., Wagner, G. (2004). A unified foundational ontology and some applications of it in business modeling. In CAiSE Workshops (3), pages 129–143.

Haase, P., & Sure, Y. (2004). D3.1.1.b State of the art on ontology evolution. *SEKT Deliverable*. Retrieved from: http://www.aifb.uni-karlsruhe.de/WBS/ysu/publications/SEKT-D3.1.1.b.pdf

Haase, P., Qi, G., (2007). An analysis of approaches to resolving inconsistencies in DL-based ontologies. Proceedings of the International Workshop on Ontology Dynamics (IWOD'07). http://kmi.open.ac.uk/events/iwod/papers/paper-13.pdf

Haase, P., Stojanovic, L. (2005). Consistent Evolution of OWL Ontologies. In A.Gomez-Perez, J. Euzenat (Eds.), LNCS, vol.3532. *The Semantic Web: Research and Applications* (pp. 182-197). Berlin, Germany: Springer. doi: 10.1007/b136731

Haase, P., Van Harmelen, F., Huang, Z., Stuckenschmidt, H., & Sure, Y. (2005). A Framework for handling inconsistency in changing ontologies. In Y.Gil, E. Motta, V. Benjamins, & M. Musen, (Eds.), LNCS: Vol. 3729. *The Semantic Web – ISWC 2005* (pp. 353-367). Berlin, Germany: Springer. doi: 10.1007/11574620

Haase, P., Völker, J., (2008). Ontology learning and reasoning – dealing with uncertainty and inconsistencies. In P. C. G. Costa, C. d'Amato, N. Fanizzi, K. B. Laskey, K. J. Laskey, T. Lukasiewicz et al. (Eds.), LNCS: Vol. 5327. *Uncertainty Reasoning for the Semantic Web I* (pp. 366-384). Berlin, Germany: Springer. doi: 10.1007/978-3-540-89765-1

Hamdi, F., Safar, B., Reynaud, C., Zargayouna, H., (2009). Partitionnement d'ontologies pour le passage à l'échelle des techniques d'alignement, EGC 2009, Revue des Nouvelles Technologies de l'Information, pp. 409-420. http://www.lri.fr/~cr/papiers/2009/EGC.pdf.

Hartmann, J., Sure, Y., Giboin, A., Maynard, D., Suarez-Figueroa, M. C., Cuel, R. (2005). Methods for ontology evaluation. Knowledge Web Deliverable D1.2.3.

Hay, D. C. (1996). Data Model Patterns. Dorset House Publishing.

Heflin, J. (2001). Towards the semantic web: Knowledge representation in a dynamic and distributed Environment. Ph.D. Thesis, University of Maryland, College Park.

Heflin, J., Hendler, J., & Luke, S. (1999). Coping with changing ontologies in a distributed environment. *Proceedings of the Workshop on Ontology Management of the 16th National Conference on Artificial Intelligence* (AAAI-99), WS-99-13, 74-79, AAAI Press.

Herrbach, C., Denise, A., Dulucq, S. (2007). Average complexity of the Jiang-Wang-Zhang pairwise tree alignment algorithm and of a RNA secondary structure alignment algorithm. Proceedings of MACIS 2007. http://www-spiral.lip6.fr/MACIS2007/Papers/submission_34.pdf

Horridge M, Parsia B, Sattler U. (2008b). Laconic and Precise Justifications in OWL". In Proceedings of the 7th International Semantic Web Conference (ISWC), Karlsruhe, Germany; LNCS 5318: 323-338.

Horridge, M., Bechhofer, S., Noppens, O. (2007). Igniting the OWL 1.1 Touch Paper: The OWL API. In Golbreich, C., Kalyanpur, A., Parsia, B., (Eds.). Proceedings of the OWLED 2007 Workshop on OWL: Experiences and Directions. Vol. 258 of CEUR Workshop Proceedings.

Horridge, M., Parsia, B., Sattler, U. (2008a). Explanation of OWL Entailments in Protégé 4, Poster et démonstration dans le cadre de la conference ISWC 2008.
http://ftp1.de.freebsd.org/Publications/CEUR-WS/Vol-401/iswc2008pd_submission_47.pdf

Horridge, M., Parsia, B., Sattler, U. (2009). Computing Explanations for Entailments in Description Logic Based Ontologies. In Automated Reasoning Workshop - Bridging the Gap between Theory and Practice, ARW09.
http://cgi.csc.liv.ac.uk/arw09/submissions/horridge_parsia_sattler_explanation_arw09.pdf

Horrocks I. & Patel-Schneider P. F. (2004). Reducing OWL Entailment to Description Logic Satisfiability. *Journal of Web Semantics*, 1(4).

Horrocks, I., Patel-Schneider, P. F., & Van Harmelen, F. (2003). From SHIQ and RDF to OWL: The Making of a Web Ontology Language. *Journal of Web Semantics*, 1(1).

Horrocks, I., Sattler, U. (2005). A Tableaux Decision Procedure for $\mathcal{SHOIQ}$. Proceedings of the 19th Joint Conference on Artificial Intelligence (IJCAI). pp: 448-453.

Horrocks, I., Sattler, U. (2007). A tableaux decision procedure for SHOIQ. Kluwer Academic Publishers, Netherlands. http://www.comlab.ox.ac.uk/people/ian.horrocks/Publications/download/2007/HoSa07a.pdf

Hunt, J. (2003). Guide to the Unified Process featuring UML, Java and Design Patterns. ISBN 978-1-85233-721-6, Springer Edition.

IEEE. (1990). Institute of Electrical and Electronics Engineers, IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. New York, USA.

Jacques, M.P., Aussenac-Gilles, N. (2006). Variabilité des performances des outils de TAL et genre textuel. Cas des patrons lexico-syntaxiques. Traitement Automatique des Langues, Association pour le Traitement Automatique des Langues (ATALA), Numéro spécial Non Thématique, Vol. 47, N. 1. http://www.atala.org/Variabilite-des-performances-des

Ji, Q., Haase, P., Qi, G. Hitzler, P. and Stadtmüller, S. (2009) RaDON - Repair and Diagnosis in Ontology Networks, The 6th Annual European Semantic Web Conference (ESWC'09) (Demo), ESWC 2009, LNCS Vol. 5554. pp: 863-867.
http://www.springerlink.com/content/g3w72l5007h07389/fulltext.pdf

Ji, Q., Qi, G., Haase, P. (2008). A relevance-based algorithm for finding justifications of DL entailments. In Technical report, University of Karlsruhe.
http://www.aifb.uni-karlsruhe.de/WBS/gqi/papers/RelAlg.pdf

Kalyanpur A, Parsia B, Cuenca-Grau B. (2006a). Beyond Asserted Axioms: Fine-Grain Justifications for OWL-DL Entailments. Proceedings de Description Logics DL2006. http://www.mindswap.org/papers/2006/beyondAxioms.pdf

Kalyanpur A, Parsia B, Sirin E, Cuenca-Grau B. (2006b). Repairing Unsatisfiable Classes in OWL Ontologies. In Proceedings of the 3rd European Semantic Web Conference (ESWC), Budva, Montenegro; LNCS Vol. 4011. pp: 170-184

Kalyanpur, A., Parsia, B., Hendler, J. (2005). A tool for working with web ontologies. *International Journal on Semantic Web and Information Systems*. Vol. 1.

Kalyanpur, A., Parsia, B., Horridge, M., Sirin, E. (2007) Finding All Justifications of OWL DL Entailments. Proceedings de ISWC/ASWC'2007. LNCS, Vol. 4825. pp: 267-280. http://iswc2007.semanticweb.org/papers/267.pdf

Kalyanpur, A., Parsia, B., Sirin, E., Grau, B.C., Hendler, J.A. (2006c). Swoop: A web ontology editing browser. *Journal of Web Semantics* 4(2), 144–153.

Klein, M. (2004). Change management for distributed ontologies. Ph.D. Thesis, Dutch Graduate School for Information and Knowledge Systems. Germany.

Klein, M., & Fensel, D., (2001). Ontology versioning on the semantic web. In I. F. Cruz, S. Decker, J. Euzenat, & D. L. McGuinness (Eds.), *Proceedings of the first International Semantic Web Working Symposium (SWWS'01)* (pp. 75-91). Stanford University, California, USA.

Klein, M., & Noy, N. F. (2003). A component-based framework for ontology evolution. In F. Giunchiglia, A. Gomez-Perez, A. Pease, H. Stuckenschmidt, Y. Sure, & S. Willmott (Eds.), CEUR

Workshop Proceeding series: Vol 71. *Proceedings of the IJCAI-2003 Workshop on Ontologies and Distributed Systems* (). Retrieved from http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-71/Klein.pdf

Klein, M., Fensel, D., Kiryakov, A., & Ognyanov, D. (2002a). Ontology versioning and change detection on the web. LNCS: Vol. 2473. *Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web* (pp. 197-212). Berlin, Germany: Springer.
doi: 10.1007/3-540-45810-7

Klein, M., Kiryakov, A., Ognyanov, D., & Fensel, D. (2002b). Finding and characterizing changes in ontologies. LNCS: Vol. 2503. *Conceptual Modeling — ER 2002* (pp. 79–89). Berlin, Germany: Springer. doi: 10.1007/3-540-45816-6

Kolp, M., Giorgini, P., Mylopoulos, J. (2003). Organizational Patterns for Early Requirements Analysis. In Proceedings of the 15th International Conference on Advanced Information Systems Engineering (CAISE'03), Velden, Austria.

Lozano-Tello, A., Gomez-Perez, A. (2004) Ontometric: A method to choose the appropriate ontology, *Journal of Database Management*, Vol. 15, N°2, p. 1-18.

Luong, P. H. (2004). Gestion de l'évolution d'un web sémantique d'entreprise. Thèse de doctorat, école doctorale sciences et technologie de l'information et de la communication, école des mines de Paris.

Luong, P.H., Dieng-Kuntz, R. (2007). A Rule-based Approach for Semantic Annotation Evolution. The Computational Intelligence Journal, 23(3):320-338. Blackwell Publishing, Malden, MA 02148, USA,.

Mäedche A, & Volz R. (2001). The ontology extraction and maintenance framework text-to-onto. *Proceedings of the Workshop on Integrating Data Mining and Knowledge Management*, at The 2001 IEEE International Conference on Data Mining ICDM'01, F. J. Kurfess, & M. Hilario, (Eds.) San Jose, California, USA.
Retrieved from http://users.csc.calpoly.edu/~fkurfess/Events/DM-KM-01/Volz.pdf

Mäedche A, Motik B, & Stojanovic L. (2003). Managing multiple and distributed ontologies in the Semantic Web. *VLDB Journal*, 12(4), 286–300. doi 10.1007/s00778-003-0102-4.

Maedche A., Staab A. (2002) Measuring similarity between ontologies. Proceedings of the European Conference on Knowledge Acquisition and Management – EKAW02. LNCS, Vol. 2473, pp. 251-263, Springer.

Mäedche, A., Stojanovic, L., Studer, R., & Volz, R. (2002). Managing multiple ontologies and ontology evolution in OntoLogging. *Proceedings of the Conference on Intelligent Information Processing* (IIP-2002), Part of the IFIP World Computer Congress WCC2002, pp. 51-63.

Maedche, A., Volz, R. (2001). The ontology Extraction & Maintenance Framework Text-To-Onto. In ICDM'01: The IEEE International Conference on Data Mining Workshop on Integrating Data Mining and Knowledge Management.

Menzis, T. (1999). Knowledge maintenance: The state-of-the-art. *The Knowledge Engineering Review*, 14(1), 1-46.

Mizoguchi R., Ikeda M. (1997). Towards Ontology Engineering. in Proceedings of the Joint Pacific Asian Conference on Expert Systems.

Moguillansky, M. O., Rotstein, N. D. and Falappa, M.A. (2008). A theoretical model to handle ontology debugging and change through argumentation. *Proceedings of the 2nd Workshop on Ontology Dynamics, (IWOD-08) at ISWC 08 Conference* (pp. 29-42). Retrieved from http://www.ics.forth.gr/~fgeo/Publications/IWOD-08_Proceedings.pdf

Muetzelfeldt, R. (2006). Declarative Modelling in Ecological and Environmental Research. Position Paper. Center of Intelligent Systems and their Applications. School of Informatics. Edinburgh University. http://www.decmod.org/documents/dmeer.pdf

Nardi, Brachman. (2002). An Introduction to Description Logics. In F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, & P.F. Patel-Schneider (Eds.), *Description Logic Handbook*, Cambridge University Press, pp. 5-44.

Noy, N. F., & Klein, M. (2003). Tracking complex changes during ontology evolution. *In Collected Posters ISWC 2003*, Sanibal Island, Florida, USA. Retrieved from http://www.stanford.edu/~natalya/papers/trackingChangesPoster.pdf

Noy, N. F., & Klein, M. (2004). Ontology evolution: Not the same as schema evolution. *Knowledge and Information Systems*, 6(4), 428-440.

Noy, N. F., & McGuinness, D. (2001) Ontology development. 101: A guide to creating your first ontology. *Stanford Knowledge Systems Laboratory Technical Report* KSL-01-05 and *Stanford Medical Informatics Technical Report* SMI-2001-0880. Retrieved from http://protege.stanford.edu/publications/ontology_development/ontology101.html

Noy, N. F., & Musen, M. A. (2003). The PROMPT suite: Interactive tools for ontology merging and mapping. *International Journal of Human-Computer Studies*, 59(6), 983-1024.

Noy, N. F., Chugh, A., Liu, W., Musen, M. A. (2006). A framework for ontology evolution in collaborative environments. LNCS: Vol. 4273. The Semantic Web - ISWC 2006 (pp. 544-558). Berlin, Germany: Springer. doi: 10.1007/11926078

Noy, N. F., Kunnatur, S., Klein, M., and Musen, M. A. (2004). Tracking changes during ontology evolution. LNCS: Vol. 3298. *The Semantic Web – ISWC 2004* (259-273). Berlin, Germany: Springer. doi: 10.1007/b102467

Noy, N., & Musen, M. (2002). PROMPTDIFF: A fixed-point algorithm for comparing ontology versions. *Proceedings of the 18th National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence.* Edmonton, Alberta, Canada: AAAI Press. ISBN 978-0-262-51129-2

O'Brien, P., & Abidi, S.S.R. (2006). Modeling intelligent ontology evolution using biological evolutionary processes. *Proceedings of the IEEE International Conference on Engineering of Intelligent Systems ICEIS 06*. Islamabad, Pakistan. doi: 0.1109/ICEIS.2006.1703172

Oberle, D., Volz, R., Motik, B., & Staab, S. (2004). An extensible ontology software environment. In S. Staab, R. Studer (Eds.), International Handbooks on Information Systems. *Handbook on Ontologies*, chapter III, pp. 311-333. Springer. Retrieved from http://www.aifb.uni-karlsruhe.de/WBS/dob/pubs/handbook2003a.pdf

Oberle. D. (2006). Semantic Management of Middleware, volume I of The Semantic Web and Beyond. Springer, New York, FEB 2006.

Oliver, D. E. Y., Shahar, M., Musen, A., & Shortliffe, E. H. (1999). Representation of change in controlled medical terminologies, *AI in Medicine*, 15(1):53–76.

OMG (Object Management Group). (2009). Ontology Definition Metamodel. Version 1.0. Chapitre 11 : The OWL Metamodel. pp: 63-92. OMG Document Number: formal/2009-05-01. http://www.omg.org/spec/ODM/1.0/PDF

Osterwalder, A. (2004). The Business Model Ontology - a proposition in a design science approach, PhD Dissertation, University of Lausanne, Switzerlan.

Pain, N. (2009). Signification et définition. Introduction de recherche. mikolka.store.googlepages.com/Significationetdefinition.pdf

Parsia, B., Sirin, E., & Kalyanpur, A. (2005). Debugging OWL ontologies. Proceedings of the 14th International World Wide Web Conference (WWW2005). pp: 633-640. http://www2005.org/cdrom/docs/p633.pdf

Patel C., Supekar K., Lee Y., Park E.K. (2004) OntoKhoj: a semantic web portal for ontology searching, ranking and classification, ACM Web Information and Data Management.

Persson, A., Stirna J. (2002). Creating an Organisational Memory Through Integration of Enterprise Modelling, Patterns, and Hypermedia: The HyperKnowledge Approach, In 11th International Conference on Information Systems Development (ISD2002), Riga, Latvia, Kluwer.

Petasis, B., Karkaletsis, V., & Paliouras, G. (2007). D4.3: Ontology Population and Enrichment: State of the Art. *BOEMIE Deliverable*. Retrieved from http://www.boemie.org/sites/default/files/D%204.3.pdf

Pinto,H., Staab, S., Tempich, C. (2004). Diligent: Towards a fine-grained methodology for distributed, loosely-controlled and evolving engineering of ontologies. In Proceedings of the 16th European Conference on Artificial Intelligence (ECAI 2004), Valencia, Spain.

Plessers, P, & De Troyer O. (2005). Ontology change detection using a version log, In Y.Gil, E. Motta, V.R. Benjamins, & M. Musen (Eds.), LNCS: Vol. 3729. *The Semantic Web – ISWC 2005* (pp. 578-592). Berlin, Germany: Springer-Verlag. doi: 10.1007/11574620

Plessers, P. (2006). An approach to web-based ontology evolution, Ph.D. Thesis, University of Brussels, Belgium.

Plessers, P., & De Troyer, O. (2006) Resolving inconsistencies in evolving ontologies. In Y. Sure, & J. Domingue (Eds.), LNCS: Vol.4011. *The Semantic Web: Research and Applications, Proceedings of the 3rd European Semantic Web Conference ESWC 2006* (pp. 200-214). Berlin, Germany: Springer. doi: 10.1007/11762256

Plessers, P., De Troyer, O., & Casteleyn, S. (2007). Understanding ontology evolution: A change detection approach. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, Elsevier Publication, 5, 39-49. Retrieved from http://www.sciencedirect.com.

Porzel R., Malaka R. (2004), A task-based approach for ontology evaluation, Proceedings of ECAI 2004 Workshop on Ontology Learning and Population.

Presutti, V., Gangemi, A. (2008). Content Ontology Design Patterns as Practical Building Blocks for Web Ontologies. In Proceedings of the 27th International Conference on Conceptual Modeling (ER 2008), Berlin, Springer.

Presutti, V., Gangemi, A., David, S., Aguado de Cea, G., Suárez-Figueroa, M.C., Montiel-Ponsoda, E., Poveda, M. (2008). A Library of Ontology Design Patterns: reusable solutions for collaborative design of networked ontologies. Deliverable D2.5.1, NeOn project.

Puppe, F. (2000). Knowledge Formalization Patterns. In Proceedings of PKAW 2000, Sydney, Australia.

Qi, G., Liu, W., Bell, D. (2006). A revision-based approach to handling inconsistency in description logics. Journal of Artificial Intelligence Review, 26(1-2): 115-128.
http://www.aifb.uni-karlsruhe.de/WBS/gqi/papers/qlb07aireview.pdf

Rector, A., Rogers, J. (2004). Patterns, properties and minimizing commitment: Reconstruction of the Galen upper ontology in OWL. In A. Gangemi and S. Borgo (Eds.), EKAW'04 Workshop on Core Ontologies in Ontology Engineering. CEUR.

Roddick, J.F. (1996). A survey of schema versioning issues for database systems. *Information and Software Technology*, 37(7), 383-393.

Scharffe, F. (2009). Correspondence Patterns Representation. PhD Thesis, Faculty of Mathematics, Computer Science and Physics of the University of Innsbruck, http://www.scharffe.fr/pub/phd-thesis/manuscript.pdf

Schlobach, S. (2005). Debugging and Semantic Clarification by Pinpointing. ESWC 2005. LNCS Vol. 3532, pp: 226-240. http://www.few.vu.nl/~schlobac/eswc05.pdf

Schlobach, S., Cornet, R. (2003). Non-Standard Reasoning Services for the Debugging of Description Logic Terminologies. Proceedings de International Joint Conferences on Artificial Intelligence IJCAI 2003. pp 355-362. http://dli.iiit.ac.in/ijcai/IJCAI-2003/PDF/053.pdf

Shaw, M. (1996). Some Patterns for Software Architectures. In J. M. Vlissides, J. O. Coplien, and N. L. Kerth, Eds., Pattern Languages of Program Design, volume 2, pages 255–269. Addison-Wesley.

Sirin E., Parsia B., Cuenca Grau B., Kalyanpur A. & Katz Y. (2007). Pellet: A practical OWL DL reasoner. *Journal of Web Semantics*, 5(2):51-53. http://pellet.owldl.com/papers/sirin07pellet.pdf

Sirin, E., Parsia, B. (2004). Pellet: An OWL DL reasoner. In V. Haaslev, & R. Moller (Eds.), CEUR Workshop Proceedings: Vol. 104, Proceedings of the International Workshop on Description Logics (DL2004).
http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-104/30Sirin-Parsia.pdf

Sotnykova, A., Vangenot, C., Cullot, N., Bennacer, N., Aufaure, M-A. (2005). Semantic Mappings in Description Logics for Database Schema Integration. In S. Spaccapietra (Ed.), *Journal on Data Semantics* III. LNCS, Vol. 3534, pp 143-176, Springer-Verlag.

Spaccapietra, S. (2005). Report on modularization of ontologies. Knowledge Web Deliverable D2.1.3.1.

Staab, S., Erdmann, M., Maedche, A. (2001). Engineering Ontologies using Semantic Patterns. In D. O'Leary and A. Preece Eds. Proceedings of the IJCAI-01 Workshop on E-business & The Intelligent Web, Seattle.

Stirna, J., Persson, A. (2002). Report of Dissemination Activities, deliverable D10, IST Programme project HyperKnowledge -- Hypermedia and Pattern Based Knowledge Management for Smart Organisations, project no. IST-2000-28401, Department of Computer and Systems Sciences, Royal Institute of Technology, Stockholm, Sweden.

Stojanovic L, Stojanovic N, Gonzalez J, & Studer R. (2003a). OntoManager— A System for the usage-based ontology management. LNCS: Vol. 2888, On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE (pp 858–875). Berlin, Germany: Springer. 10.1007/b94348

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

Stojanovic, L., & Motik, B. (2002). Ontology evolution within ontology editors. CEUR-WS: Vol. 62, Proceedings *of the OntoWeb-SIG3 Workshop Evaluation of Ontology-based Tools (EON2002) at the 13th International Conference on Knowledge Engineering and Knowledge Management* (EKAW 2002) (pp. 53-62). Retrieved from http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-62/EON2002_Stojanovic.pdf

Stojanovic, L., Maedche, A., Motik, B., & Stojanovic, N. (2002a). User-driven ontology evolution management. LNCS: Vol. 2473, *Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web, Proceedings of the 13th International Conference* EKAW2002 (pp. 285-300). Berlin, Germany: Springer-Verlag. doi: 10.1007/3-540-45810-7

Stojanovic, L., Maedche, M., Stojanovic, N., & Studer, R. (2003b). Ontology evolution as reconfiguration-design problem Solving. *Proceedings of the 2nd international conference on Knowledge capture K-CAP'03* (pp.162-171). New York, USA: ACM. ISBN:1-58113-583-1

Stojanovic, L., Stojanovic, N., & Handschuh, S. (2002b). Evolution of the metadata in the ontology-based knowledge management systems. LNI: Vol.10 GI 2002, *Proceedings of the 1st German Workshop on Experience Management GWEM 2002 (*pp. 65-77).

Stojanovic, N., Stojanovic, L., & Volz, R. (2002c). A Reverse engineering approach for migrating data-intensive web sites to the semantic web. *Proceedings of the Conference on Intelligent Information Processing* (IIP-2002), Part of the IFIP World Computer Congress WCC2002, (pp.141-154). Montreal, Canada.

Suntisrivaraporn, B., Qi, G., Ji, Q., Haase, P. (2008) A Modularization-based Approach to Finding All Justifications for OWL DL Entailments. ASWC 2008. http://www.aifb.uni-karlsruhe.de/WBS/pha/publications/justifications08aswc.pdf

Supekar, K. (2005) A peer-review approach for ontology evaluation, Proceedings of the 8th International. Protégé Conference.

Sure, Y. (2002). On-to-knowledge – ontology based knowledge management tools and their application. *German Journal Kuenstliche Intelligenz*, Special Issue on Knowledge Management, 35-37.

Sure, Y., Studer, (2002). On-to-Knowledge methodology, In Davies, Fensel, Van Harmelen Eds., *Onto-Knowledge : Semantic Web enabled Knowledge Management*, chapitre 3, pp 33-46, Wiley & Sons.

Svatek V. (2004). Design patterns for semantic web ontologies: Motivation and discussion. 7ème conférence Business Information Systems, Poznan.

Tallis, M., Gill, Y. (1999). Designing Scripts to Guide Users in Modifying Knowledge-based Systems, *AAAI/IAAI*, pp : 242-249.

Tartir, S., Arpinar, I. B. (2007) Ontology Evaluation and Ranking using OntoQA. In the proceedings of the International Conference on Semantic Computing ICSC 2007, pp. 185--192. IEEE Computer Society. http://www.cs.uga.edu/~budak/papers/OntoQA-ICSC-2007.pdf

Tartir, S., Arpinar, I. B., Moore, M., Sheth, A. P., Aleman-Meza. B. (2005) OntoQA: Metric-based ontology quality analysis. In Proceedings of IEEE Workshop on Knowledge Acquisition from Distributed, Autonomous, Semantically Heterogeneous Data and Knowledge Sources, pp. 45--53. http://www.cs.uga.edu/~budak/papers/ontoqa.pdf

Tempich, C. Pinto, H.S., Sure, Y. Staab, S., (2005). An argumentation ontology for Distributed, Loosely-controlled and evolvInG engineering processes of oNTologies (DILIGENT). In Proceedings of the European semantic web conference, ESWC05, LNCS, Vol. 3532, pp 241-256.

Uschold, M., King, M. (1995). Towards a Methodology for Building Ontologies. In Proceedings of the Workshop on Basic Ontological Issues in Knowledge Sharing at the International Joint Conference on Artificial Intelligence (IJCAI'1995).

Van Harmelen, F., Horrocks, I. (2000). FAQs on OIL: the Ontology Inference Layer. *IEEE Intelligent Systems*, Vol 15(6), pp. 69-72.

Völker, J., Haase, P., Hitzler, P. (2008) Learning Expressive Ontologies. In P. Buitelaar, P. Cimiano, Ontology Learning and Population: Bridging the Gap between Text and Knowledge. *Frontiers in Artificial Intelligence and Applications*. Vol. 167, pp. 45-69. IOS Press, Amsterdam. ISBN: 9781586038182. http://www.aifb.uni-karlsruhe.de/WBS/jvo/publications/leo-chapter_2008.pdf

Völker, J., Hitzler, P., & Cimiano, P., (2007b). Acquisition of OWL DL axioms from lexical resources. In E. Franconi, M. Kifer, & W. May (Eds.), LNCS: Vol. 4519. *The Semantic Web: Research and Applications, Proceedings of the 6th International Semantic Web Conference, ISWC 2007* (pp. 670-685). Berlin, Germany: Springer. doi: 10.1007/978-3-540-72667-8
Retrieved from http://www.eswc2007.org/pdf/eswc07-voelker2.pdf

Volker, J., Vrandecic, D., Sure, Y. (2005) Automatic evaluation of ontologies (AEON). In Y. Gil, E. Motta, V. R. Benjamins, and M. A. Musen, Eds. Proceedings of the 4th International Semantic Web Conference (ISWC2005). LNCS, Vol. 3729, pp: 716–731. Springer Verlag Berlin-Heidelberg. doi: 10.1007/11574620

Völker, J., Vrandecic, D., Sure, Y., & Hotho, A., (2007a). Learning disjointness. In E. Franconi, M. Kifer, & W. May (Eds.), LNCS: Vol. 4519. *The Semantic Web: Research and Applications, Proceedings of the 6th International Semantic Web Conference, ISWC 2007* (pp. 175-189). Berlin, Germany: Springer. doi: 10.1007/978-3-540-72667-8
Retrieved from http://www.eswc2007.org/pdf/eswc07-voelker1.pdf

Vrandecic, D., Sure, Y. (2007) How to Design Better Ontology Metrics. In Proceedings of ESWC 2007. LNCS, vol 4519 pp 311-325. Springer. http://kmi.open.ac.uk/events/eswc07/pdf/eswc07-vrandecic.pdf

Vrandecic, D., Surez-Figueroa, M. del C., Gangemi, A., Sure, Y., Eds. (2006) Proceedings of the 4th International Workshop on Evaluation of Ontologies for the Web (EON2006) at the 15th International World Wide Web Conference (WWW 2006).

Wang, H., Horridge, M., Rector, A., Drummond, N., & Seidenberg, J. (2005). Debugging OWL-DL ontologies: A heuristic approach. In Y. Gil, E. Motta, V. R. Benjamins, & M. A. Musen (Eds.), LNCS: Vol. 3729. *The Semantic Web – ISWC 2005* (pp. 745-757). Berlin, Germany: Springer. doi: 10.1007/11574620

Welty, C., Andersen, W. (2005) Towards OntoClean 2.0: a framework for rigidity. *Journal of Applied Ontology*. 1(1):107-116. Amsterdam:IOS Press.
http://domino.research.ibm.com/library/cyberdig.nsf/papers/0511B80E70CADF09852570A4004FBDA5/$File/rc23754.pdf

Xuan D. N., Bellatreche L. & Pierra G. (2006). A versioning management model for ontology-based data warehouses, *Proceedings of 8th International Conference on Data Warehousing and Knowledge Discovery DaWak'06*. Krakow, Poland
Retrieved from http://www.lisi.ensma.fr/ftp/pub/documents/papers/2006/2006-TICDWKD-XUAN.pdf

Yang, Z., Zhan, D., Ye, C. (2006) Evaluation Metrics for Ontology Complexity and Evaluation Analysis, IEEE International Conference on e-Business Engineering (ICEBE06).

Annexes

Annexe A : Apprentissage d'ontologies

L'une des utilisations des ontologies est de faciliter le processus d'acquisition de connaissances. Toutefois, pour construire une ontologie ou affiner une ontologie existante, il est nécessaire d'appliquer un processus d'extraction de connaissances. Pour automatiser partiellement ce processus, plusieurs approches d'apprentissage ont été proposées. Elles se basent sur des techniques d'analyse de langages naturels et des techniques d'apprentissage par la machine.

Dans (Maedche & Staab, 2001) les approches d'apprentissage sont classées selon les types de données en entrée : des sources textuelles, des documents Web, des sources semi-structurées, des schémas relationnels, des bases de connaissances, ou encore une combinaison de sources hétérogènes. Pour chacune de ces classes, différentes techniques d'apprentissage peuvent être appliquées : des techniques linguistiques (analyse syntaxique et distributionnelle, patrons lexico-syntaxique), des techniques de fouilles de textes (classification symbolique, classification numérique) ou des techniques hybrides.

Dans ce qui suit, nous présentons succinctement les différents types de techniques d'apprentissage ainsi que les principales approches existantes en fonction des sources en entrée (sans prétendre à une étude exhaustive).

I- TECHNIQUES D'APPRENTISSAGE

I.1- Techniques linguistiques

Les analyses linguistiques (lexicales et syntaxiques) se basent sur des marqueurs linguistiques pour sélectionner des segments textuels qui peuvent correspondre à des termes complexes (syntagmes nominaux composés de suites de catégories grammaticales régulières). Certaines de ces techniques repèrent des séquences de mots par projection de modèles prédéfinis (patrons) et appliquent une série de règles de découpage. D'autres techniques appliquent également, des règles de transformation en se guidant de connaissances extérieures (thésaurus).

Les techniques linguistiques sont généralement accompagnées d'analyses statistiques et sont utilisées pour l'extraction de concepts et de relations taxonomiques et non taxonomiques à partir de textes.

Les analyses statistiques s'appuient essentiellement sur le principe suivant : « si un mot X apparaît plus fréquemment dans l'entourage d'un mot Y qu'ailleurs dans le texte, alors X et Y forment une combinaison significative ». L'entourage est le contexte de cette association de mots (X, Y) appelée cooccurrence de X et Y. Il existe trois approches statistiques fondées sur la notion de cooccurrence :

- L'approche distributionnelle de Harris dont le but est de découvrir les relations entre termes en fonction de leur régularité d'apparition commune.
- Une approche dont la ressemblance des contextes définit la relation de cooccurrence entre les termes. L'étude du recouvrement des environnements sémantiques permet d'établir des relations entre les termes (y compris s'ils ne co-occurrent pas).
- Une approche de calcul de motifs fréquents qui se base sur l'apparition de chaines de séquences se répétant dans un même texte. Une fréquence minimale d'apparition dans le texte est fixée afin d'éliminer les faibles occurrences. Pour chaque motif, l'ensemble des suites de termes dans le texte commençant par cette forme est répertorié. Le processus est réitéré pour chaque motif.

Parmi les travaux qui combinent les analyses présentées ci-dessus, nous pouvons citer l'approche de Bourigault et al. (Bourigault & Fabre, 2000), (Bourigault et al., 2005) qui consiste à étiqueter le corpus, l'analyser syntaxiquement et appliquer ensuite une analyse distributionnelle pour rapprocher les couples d'unités sur la base de contextes syntaxiquement identiques et induire des classes sémantiques de mots. L'identification des relations continue en projetant des patrons lexico-syntaxiques sur le texte analysé et en élaborant des études statistiques sur les termes et les formes syntaxiques identifiées. L'identification des relations non taxonomiques se base sur une recherche de motifs fréquents à partir des patrons. La difficulté d'extraction de relations non-taxonomiques réside dans la labellisation des relations après leur découverte.

I.2- Techniques de fouille de textes

Ces techniques appliquées sur des textes et des thésaurus (tels que Wordnet), sont utilisées pour la construction, l'enrichissement et le peuplement d'ontologies. L'enrichissement d'ontologie se fait par la découverte de nouvelles relations entre concepts d'une ontologie existante (apprendre des relations d'hyponymie et produire des patrons lexico-syntaxiques à partir de paires de concepts liés, extraits de l'ontologie) (Hearst, 1998). Le peuplement d'ontologies (telles que l'ontologie *Gene Ontology*) se base sur l'extraction et la labellisation de classes de mots (entités nommées).

Les techniques de fouilles de texte regroupent deux types de classification :

- Classification symbolique ou conceptuelle (treillis de Gallois, graphe conceptuel, arbres de décision, etc.). La difficulté de cette classification réside dans le traitement de gros volume de données. Le calcul de la distance entre concepts se base sur les fonctions syntaxiques que jouent les termes désignant ces concepts dans le texte.

  La méthode de Alfonseca et al. (Alfonseca et al., 2002a) classe les concepts en se basant sur les signatures sémantiques ou contextuelles (*Topic Signature*) pour identifier les concepts partageant les mêmes contextes.
- Classification numérique (clustering par partitionnement ou clustering hiérarchique, réseaux de neurones, réseaux bayésiens, chaines de Markov, etc.). Elle s'applique sur de gros corpus et a souvent recours à des dictionnaires pour éliminer les mots de contenu vide lors de l'extraction de termes. La difficulté de cette classification réside dans la définition des mesures de similarité entre concepts. Plusieurs mesures de similarité ont été empruntées à la recherche d'information (TFIDF, l'entropie, pertinence du domaine PD, consensus de domaine CD) mais des conclusions contradictoires ont été tirées sur l'efficacité de ces mesures. La méthode *Doddle* (Sekiuchi et al., 1998) propose de nouvelles mesures de similarités pour l'extraction de concepts en se basant sur la distribution des mots dans le texte. Cette méthode construit un espace multidimensionnel basé sur l'analyse de cooccurrence, à partir duquel sont extraits les paires de concepts similaires. Par ailleurs, elle permet d'extraire des relations taxonomiques de Wordnet et des relations non taxonomiques par la recherche de règles d'association.

  En général, les relations taxonomiques sont extraites en se basant sur l'analyse des occurrences et le regroupement hiérarchique des termes (méthodes hiérarchiques ascendantes ou descendantes) ou sur des mesures probabilistes. Les relations non taxonomiques sont extraites en se basant sur les termes centraux (noms et noms propres apparaissant fréquemment) et les règles d'association.

  Dans (Karoui et al., 2006), une méthode d'apprentissage d'ontologies de domaine à partir du web est proposée. Cette méthode se base sur la structure des pages web pour définir une hiérarchie contextuelle sans faire intervenir de connaissances à priori. Une étape de prétraitement permet de sélectionner les termes pertinents à classer. Une pondération est affectée aux termes sélectionnés en fonction de leur position dans l'hiérarchie conceptuelle. Les termes sont alors classés automatiquement pour permettre l'extraction des concepts.

I.3- Techniques hybrides

Les techniques linguistiques et de fouilles de texte sont complémentaires et peuvent être combinées pour construire ou enrichir des ontologies (Faure et Nedellec, 1998) (Khan et Luo, 2002).

Dans (Ben Mustapha et al., 2007), une approche incrémentale d'apprentissage d'ontologie de domaine pour le web sémantique, basée sur l'utilisation d'une méta-ontologie, est proposée. Les concepts sont représentés par un vecteur d'expressions nominales qui le référencent, pondérées par leur fréquence d'apparition dans le corpus. Les relations sont représentées par un ensemble de cadres syntaxiques et de patrons lexico-syntaxiques pondérés par leur fréquence d'utilisation. Le corpus est mis-à jour en vue de corriger les erreurs dues à la fréquence des patrons.

La méthode tient compte aussi des axiomes : les axiomes d'apprentissage qui définissent les règles d'apprentissage dans la méta-ontologie (règles d'extraction de concepts, relations taxonomiques et non taxonomiques à partir du contenu textuel des pages Web) et des axiomes de domaine en appliquant des règles d'association. Les axiomes de domaine expriment des restrictions sur les relations : les axiomes génériques expriment des restrictions générales entre les concepts du domaine, les axiomes spécifiques expriment des contraintes relatives aux instances. Les axiomes sont formalisés en logique de premier ordre (prémisses, conséquences).

La spécification des connaissances servant à l'apprentissage des ontologies de domaine se base sur des techniques syntaxiques (l'utilisation de la grammaire, apprentissage de patrons de phrases), les méthodes de surface (les marqueurs de frontières), des patrons lexico-syntaxiques (définition et apprentissage de patrons d'hyponymie), des techniques statistiques (sélection des termes candidats en fonction de leurs occurrences dans les documents, suppression des mots vides, signature sémantique et cooccurrence des verbes), des techniques de classification conceptuelle, des techniques de classification numérique (regroupement hiérarchique et calcul des probabilités des cooccurrences) et les règles d'association.

II- APPROCHES D'APPRENTISSAGE SELON LES SOURCES EN ENTREE

II.1- Apprentissage d'ontologie à partir de dictionnaires

La plupart des méthodes d'apprentissage d'ontologie à partir de dictionnaires, utilisent Wordnet comme référence pour apprendre des concepts et des relations (Hearst, 1992), (Jannik, 1999) (Moldovan & Girju, 2000). Elles se basent sur des analyses linguistiques et sémantiques.

II.2- Apprentissage d'ontologie à partir de documents Web

Plusieurs approches enrichissent des ontologies à partir de documents Web pour pallier la difficulté de collecter manuellement des textes relatifs au domaine modélisé (Aguire et al., 2000) (Faatz & Steinmetz, 2000) (Ben Mustapha et al., 2007) (Karoui et al., 2006).

Dans (Aguire et al., 2000), les auteurs proposent une méthode de clustering qui enrichit les concepts d'une ontologie en se basant sur des documents Web classés en collections à l'aide d'un corpus étiqueté et, de Wordnet. Pour chacune des collections formées, les mots et leurs fréquences respectives sont extraits et comparés aux autres collections.

Dans (Navigli & Velardi, 2004), les auteurs proposent une approche réduisant la confusion terminologique et conceptuelle entre les membres d'une communauté virtuelle, en se basant sur un apprentissage de concepts et de relations, organisé en trois phases : extraction de terminologies à partir de sites web et d'entrepôts de documents web, interprétation sémantique des termes et identification des relations taxonomiques.

Dans (Sanchez & Moreno, 2004), les auteurs proposent une méthode qui construit des ontologies uniquement à partir du web (utilise seulement un moteur de recherche sans recours à des

connaissances à priori, ni à des techniques de traitement de langage naturel, ni à l'utilisation de larges corpus ou thésaurus) en recherchant les concepts les plus pertinents d'un domaine à partir d'une large collection de sites web. Cette méthode se base sur un algorithme récursif constituée de 6 étapes (Sanchez & Moreno, 2004): définition des mots-clés du domaine, recherche de collections de sites web relatifs aux mots-clés, analyse exhaustive de chaque site, recherche des mots-clés initiaux dans un site web et des mots qui les précèdent ou succèdent (ils seront des concepts candidats), analyse statistique de chaque concept extrait (basée sur le nombre d'occurrences et autres critères) et définition de nouveaux mots-clés. Ainsi, l'ontologie est raffinée au fur et à mesure en fusionnant les concepts et sous-concepts identifiés. Cependant, seules les relations taxonomiques sont extraites par cette méthode.

II.3- Apprentissage d'ontologie à partir de schémas semi-structurés

Des approches de *Clustering* et de *reconnaissance de patrons* permettent d'extraire des connaissances ontologiques à partir de sources semi-structurées telles que des schémas XML ou RDF.

Dans (Deltei et al., 2000), les auteurs proposent une approche d'apprentissage d'ontologie à partir des annotations sémantiques d'une base de documents RDF. Cette approche se focalise sur la problématique d'enrichissement d'ontologie par des concepts définis. Les annotations sont exploitées pour repérer les concepts définis. L'approche s'applique dans le cadre d'un Web sémantique de taille limitée (une organisation par exemple).

L'approche applique un apprentissage symbolique basé sur des techniques de regroupement conceptuel (former des classes rassemblant les objets similaires). Cependant, plutôt que d'appliquer des heuristiques prédictives de classification apprise (qui ne peuvent être objectives), l'approche définit un apprentissage systématique dont le résultat sera évalué par l'expert du domaine (c'est l'utilisateur qui choisit les classes qu'il juge intéressantes). L'algorithme proposé classe les objets de façon systématique selon les motifs relationnels avec une prise en compte progressive de l'expressivité (pour gérer la complexité des descriptions relationnelles) (Deltei, 2002) : la taille des descriptions apprises augmente graduellement en incrémentant à chaque étape, le nombre maximal d'arcs ou de prédicats.

La principale particularité de l'approche réside dans la forme des données : les objets font partie d'un même graphe et n'ont pas de description propre (l'extraction d'une description particulière d'un objet se fait à partir du graphe global). Par ailleurs, la recherche de concepts tient compte de l'ontologie existante.

II.4- Apprentissage d'ontologie à partir de schémas relationnels[*]

Différents travaux se sont basés sur les données et les schémas de bases de données pour construire des ontologies (Johannesson, 1994), (Kashyap, 1999), (Rubin et al., 2002). L'apprentissage est assuré à travers un processus d'appariement basé sur un ensemble de règles pour passer d'un modèle entité-relation vers des primitives ontologiques, ou encore à travers un processus de réingénierie.

II.5- Apprentissage d'ontologie à partir de bases de connaissances

Une seule méthodologie a pu être identifiée concernant l'apprentissage à partir de bases de connaissances : c'est l'approche proposée par Suryanto et Compton (Suryanto & Compton, 2001) qui consiste à générer une ontologie à partir des règles d'une base de connaissances.

[*] Un état de l'art sur l'alignement de bases de données relationnelles à RDF du groupe W3C RDB2RDF peut être consulté sur : http://esw.w3.org/topic/Rdb2RdfXG/StateOfTheArt

II.6- Apprentissage d'ontologie à partir d'instances

Morik et Kietz proposent de partir d'instances particulières à partir de fichiers de données pour générer des ontologies (Morik & Kietz, 1989).

II.7- Apprentissage d'ontologie à partir de texte

Les principales méthodes d'apprentissage d'ontologie appliquent des techniques d'apprentissage à partir de textes. Notons que les concepts et les relations qui décrivent les connaissances ontologiques se situent à un niveau conceptuel alors que les textes utilisés pour l'apprentissage de ces connaissances se situent à un niveau linguistique. Ainsi, l'objectif des approches d'apprentissage est de projeter les structures du niveau linguistique à un niveau conceptuel (Gomez-Perez et al., 2004).

II.7.1- Techniques d'apprentissage à partir de textes

Les techniques d'apprentissage à partir de texte sont classées comme suit[†] :

- Les techniques d'extraction basées sur des *patrons ou modèles* (Hearst, 1992), (Morin, 1999): Ces techniques permettent de ressortir des relations conceptuelles à partir de séquences de mots du texte en se basant sur des patrons. Les linguistes définissent généralement les patrons de façon manuelle. Pour pallier le problème de spécification de patrons lexico-linguistiques, des méthodes d'extraction automatique de patrons ont été définies (Hearst, 1998). Elles se basent sur l'étude des régularités syntaxiques entre deux concepts donnés en observant la réalisation d'une relation – existante dans l'ontologie – dans le corpus et en schématisant le contexte lexical et syntaxique afin de constituer des patrons (des patrons liés à l'hypéronymie pour mettre en relation des couples père-fils). Cependant, les patrons appris peuvent présenter une marge d'erreurs. Dans [Alfonseca & Mandahar, 2002b], les auteurs proposent de combiner ces méthodes avec les signatures contextuelles ou sémantiques (*Topic Signature*) afin d'atténuer les erreurs. Ces signatures sont construites en collectant pour chaque concept, l'ensemble des concepts dont leur fréquences d'occurrence ou leur poids respectifs sont remarquables.
- Les techniques basées sur les *règles d'association* (Skrikant & Agrawal, 1995): Introduites dans le contexte des bases de données pour déduire des résultats de transaction, les règles d'association ont été appliquées en datamining, pour la découverte d'informations contenues dans les bases de données. Dans le cadre des ontologies, elles sont utilisées pour la découverte de relations non-taxonomiques entre concepts en se basant sur l'hiérarchie de concepts et les statistiques des cooccurrences des termes dans le texte (Maedche & Staab, 2000a).
- *Regroupement* de concepts (clustering conceptuel ou numérique) (Michalsky, 1980) (Faure & Poibeau, 2000) (Sekiuchi et al., 1998): Ces techniques partent d'un ensemble de concepts et tentent de les regrouper en se basant sur les distances sémantiques entre ces concepts (une distance seuil est définie pour décider du regroupement).
- L'*élagage* d'ontologie (Kietz et al., 2000): L'élagage permet de définir une ontologie de domaine à partir d'une ontologie générique et d'un ensemble de corpus textuels représentatifs du domaine.
- L'*apprentissage* de concepts (Hahn & Schulz, 2000): Il consiste à enrichir une taxonomie de concepts existante par acquisition de nouveaux concepts à partir de textes représentant le monde réel. L'apprentissage de concepts combine les différentes techniques précédemment citées (regroupement de concepts, extraction basée sur des patrons, etc.)

[†] Cette classification est en partie prise de (Maedche & Staab, 2000).

II.7.2- Méthodes d'apprentissage d'ontologies à partir de textes

La méthode de Maedche et al. Cette méthode construit des ontologies en appliquant des techniques d'analyse de langage naturel qui utilisent les patrons lexico-syntaxiques (Kietz et al., 2000), le clustering conceptuel, l'élagage d'ontologie en se basant sur une ontologie générique telle que SENSUS et l'apprentissage de concepts (Maedche et Staab, 2000b). Elle s'articule en cinq activités : la sélection des sources, l'apprentissage de concepts, le centrage du domaine, l'apprentissage de relations et l'évaluation (pour décider de la continuité du processus).

La méthode de Aussenac-Gilles et al. Cette méthode a pour objectif d'élaborer un modèle conceptuel d'un domaine par analyse de corpus de documents techniques relatifs à ce domaine (Aussenac-Gilles et al., 2000a) (Aussenac-Gilles et al., 2000b). Elle offre un guidage à la modélisation d'ontologies à partir de textes en se basant sur l'utilisation de patrons et le regroupement de concepts. Les patrons tiennent comptent aussi bien des verbes que des expressions composées en analysant les termes qui précèdent les concepts pour découvrir de nouvelles relations non taxonomiques (Aussenac-Gilles et al. 2000a).

La définition des patrons lexico-syntaxiques se fait de manière manuelle ce qui nécessite une validation humaine. Les patrons définis sont de haute qualité essentiellement pour les relations génériques comme l'hyperonymie, mais ne sont pas assez fréquents dans les corpus c'est pourquoi, la productivité des relations apprises reste relativement faible (Aussenac-Gilles et al., 2000b).

Le processus d'apprentissage suppose que le constructeur de l'ontologie est expert du domaine (pour sélectionner les termes du domaine et les assigner aux concepts et relations) et qu'il tient compte de l'objectif applicatif de l'ontologie à construire (Aussenac-Gilles et al., 2002).

La méthode est organisée en quatre activités : la constitution du corpus, l'étude linguistique (extraire les termes, les relations lexicales et les groupes de synonymes), la normalisation (construire un modèle conceptuel sous forme de réseau sémantique) à deux niveaux linguistique puis conceptuel, et une activité de formalisation (implémenter le réseau sémantique dans un langage formel).

La méthode de Nobécourt (Nobécourt, 2000) est constituée d'une phase de modélisation et d'une phase de représentation. Elle se base sur les techniques linguistiques et les patrons lexico-syntaxiques.

Le système KAT (Knowledge Acquisition from Text) (Moldovan & Girju, 2000) propose une méthode en quatre étapes : apprentissage de nouveaux concepts à partir de texte, classification des concepts en se basant sur l'analyse des expressions associées aux concepts candidats, apprentissage de relations à partir de patrons, et évaluation de l'ontologie.

La méthode ASIUM (Faure et Nedellec, 1998) combine des techniques linguistiques et de clustering en vue d'apprendre des cadres de catégorisation en s'appuyant sur une technique de regroupement de concepts. La méthode se base sur deux algorithmes « Asium-Best » et « Asium-Level ». Le premier algorithme – itératif - permet de calculer les distances entre des paires de clusters et les comparer à un seuil, et d'agréger les paires de clusters pour former de nouveaux clusters (lorsque la distance est inférieure au seuil). Le second algorithme permet de construire l'ontologie graduellement (niveau par niveau). Cette technique de clustering conceptuel permet d'apprendre de nouvelles relations non taxonomiques.

L'approche de Khan et Luo (Khan & Luo, 2002) définit deux algorithmes combinant classification hiérarchique et des techniques linguistiques pour construire des ontologies à partir de textes.

III- SYNTHESE

Différentes techniques d'apprentissage existent. Ces techniques sont complémentaires et peuvent être combinées. Le choix des techniques varie en fonction des sources en entrée et du type des connaissances à extraire (Table 1).

	Concepts	Relations taxonomiques	Relations non taxonomiques	Axiomes	Instances
Sources textuelles, Web, thésaurus	▪ Techniques d'analyse linguistiques combinées à des techniques statistiques. En résumé le processus consiste à étiqueter le corpus, l'analyser syntaxiquement puis appliquer une analyse statistique (distributionnelle par exemple) pour rapprocher les couples d'unités sur la base de contextes syntaxiquement identiques et induire les classes sémantiques de mots.	▪ Techniques d'analyse linguistiques combinées à des techniques statistiques. Le procédé d'extraction de concept est enrichi par la projection de patrons lexico-syntaxiques sur le texte analysé et l'analyse statistique des termes et des formes syntaxiques identifiées.	▪ Techniques d'analyse linguistiques combinées à des techniques statistiques. L'identification se fait essentiellement en appliquant une approche statistique de recherche de motifs fréquents à partir des patrons. La difficulté réside dans la labellisation des relations découvertes.		▪ Techniques d'analyse linguistiques pour le peuplement d'ontologie.
	▪ Techniques de fouille de textes (classification symbolique ou conceptuelle et classification numérique).	▪ Techniques de fouille de textes par découverte de nouvelles relations entre les concepts d'une ontologie existante. Apprendre des relations d'hyponymie et produire des patrons à partir de paires de concepts liés (classification symbolique ou conceptuelle et classification numérique).	▪ Techniques de fouille de textes par découverte de nouvelles relations entre les concepts d'une ontologie existante (classification numérique et recherche de relations non taxonomiques en se basant sur les termes centraux (noms et noms propres fréquents) et en appliquant des règles d'association (hiérarchie de concepts et statistiques des cooccurrences des termes)).		▪ Techniques de fouille de textes pour le peuplement d'ontologies en se basant sur l'extraction et la labellisation de classes de d'entités nommées.

	▪ Techniques hybrides.	▪ Techniques hybrides.	▪ Techniques hybrides.	▪ Techniques hybrides.	▪ Techniques hybrides.
Ontologie générique et corpus textuel	▪ Elagage d'ontologie pour définir une ontologie de domaine à partir d'une ontologie générique et d'un ensemble de corpus représentatif.	▪ Elagage d'ontologie pour définir une ontologie de domaine à partir d'une ontologie générique et d'un ensemble de corpus représentatif.	▪ Elagage d'ontologie pour définir une ontologie de domaine à partir d'une ontologie générique et d'un ensemble de corpus représentatif.		
Schémas semi-structurés (XML, RDF)	▪ Techniques de clustering symbolique (regroupement conceptuel) et de reconnaissance de Patrons. Appliquées aussi pour l'enrichissement d'ontologie par des concepts définis (mais dans un cadre de web sémantique de taille limitée).	▪ Techniques de clustering symbolique (regroupement conceptuel) et de reconnaissance de Patrons.	▪ Techniques de clustering symbolique (regroupement conceptuel) et de reconnaissance de Patrons.		
Schémas relationnels	▪ Processus d'appariement basé sur des règles de passage d'un modèle entité-relation vers des primitives ontologiques.	▪ Processus d'appariement basé sur des règles de passage d'un modèle entité-relation vers des primitives ontologiques.	▪ Processus d'appariement basé sur des règles de passage d'un modèle entité-relation vers des primitives ontologiques.		**Base de données en entrée** ▪ Processus d'appariement données / instances.
	▪ Processus de réingénierie.	▪ Processus de réingénierie.	▪ Processus de réingénierie.		

Table 1. Classement des techniques d'apprentissage d'ontologies par rapport au type des connaissances extraites et des sources en entrée.

RÉFÉRENCES

Aguire E., Ansa O., Hovy E. et Martinez D. (2000). Enriching very large ontologies using the WWW. In Proceedings of Ontology Construction Workshop of the European Conference ECAI-00.

Alfonseca E., Manandhar S. (2002a). An unsupervised method for general named entity recognition and automated concept discovery. Proceedings of the 1st International Conference en General Wordnet.

Alfonseca E., Mandahar S. (2002b). Improving Ontology Refinement Method with Hyponymy Patterns. Language Resources and Evaluation (LREC).

Aussenac-Gilles N., Biébow B., Szulman S. (2000a). Revisiting Ontology Design : a methodology Based on Corpus Analysis, In Dieng R, Corby O. (eds) 12th International Conference in Knowledge engineering and Knowledge Management EKAW'00, (LNAI 1937) Springer-Verlag, Berlin, Germany, pp 172-188.

Aussenac-Gilles N., Biébow B., Szulman S. (2000b). Corpus Analysis for conceptual modelling. In Aussenac-Gilles N, Biébow B. et Szulman S. (eds) EKAW'00 Workshop on Ontologies and Texts, France. CEUR Workshop Proceedings http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-51/.

Aussenac-Gilles N., Biébow B., Szulman S. (2002). Modelling the travelling domain from NLP description with TERMINEA. In Angele J, Sure Y, EKAW'02 Workshop on Evaluation of Ontology-based Tools (EON2002), CEUR Worshop Proceedings 62:112-128. http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-62/.

Ben Mustapha N., Baazaoui H., Aufaure M-A. (2007). A prototype for Knowledge Extraction from Semantic Web based on Ontological Components Construction. 3rd International Conference on Web Information Systems and Technologies (WEBIST).

Bourigault, D., Fabre, C. (2000). Approche linguistique pour l'analyse syntaxique de corpus". Cahiers de Grammaire, 25:131-151, Sémantique et corpus, coordonné par Anne Condamines, ERSS - UMR 5610, CNRS et Université de Toulouse-Le Mirail.

Bourigault, D., Fabre, C., Frérot, C., Jacques, M.-P., Ozdowska, S. (2005). Syntex, un analyseur syntaxique de corpus. Actes du colloque TALN, 6-10 juin 2005, atelier EASy : campagne d'évaluation des analyseurs syntaxiques, Vol. II, pp. 17-20.

Deltei A. Faron C., Dieng R. (2000). Learning ontologies from RDF annotations. Proceedings of the IJCAI Workshop on the Web and Databases.

Deltei A., (2002). Représentation et apprentissage de concepts et d'ontologies pour le Web sémantique. Thèse en Sciences, discipline Informatique, Université Nice-Sophia Antipolis.

Faatz A., Steinmetz R. (2000). Ontology enrichment with texts from the WWW. Semantic Web mining 2sd Workshop at ECML/PKDD-02.

Faure D., Nedellec, C. (1998). A corpus-based conceptual clustering method for web frames and ontology acquisition. LREC workshop on adapting lexical and corpus resources to sublanguages and applications.

Faure D., Poibeau T. (2000). First experiments of using semantic knowledge learning by ASIUM for information extraction task using INTEX. In Staab S, Maedche A. Nedellec C, Wiemer-Hastings P (Eds.) Ontology Learning ECAI-2000 Workshop pp 7-12.

Gomez-Perez, Fernandez-Lopez & Corcho, (2004). Ontological Engineering". Springer, Advenced Information and Knowledge Processing.

Hahn U., Schulz S. (2000). Towards Very large Terminological Knowledge Bases: A Case Study from Medicine. Hamilton HJ (Ed.) Advances in Artificial Intelligence, 13th Biennial Conference of Canadian Society for Computational Studies of Intelligence (AI 2000), (LNCS 1822) Springer-Verlag, Berlin, Germany, pp 176-186.

Hearst MA. (1992). Automatic acquisition of hyponyms from large text corpora. In Zampolli A. (ed) Proceedings of the 15th International Conference on Computational Linguistic (COLING-92)

Hearst M.A. (1998). Automated Discovery of Wordnet Relations. Wordnet an Electronic Lexical Database. MIT Press, Cambridge, MA, pp 132-152.

Jannik J. (1999). Thesaurus Entry Extraction from an on-line Dictionary. Proceedings of Fusion'99

Johannesson P. (1994). A Method for Transforming Relational Schemas into Conceptual Schemas. 10th International Conference on Data Engineering, M. Rusinkiewicz (Ed.), pp:115 – 122.

Karoui L., Aufaure M.-A., et Bennacer N. (2006). Context-based Hierachical Clustering for the Ontology Learning, the 2006 IEEE/WIC/ACM International Conference on Web Intelligence (WI-06) jointly with the 2006 IEEE/WIC/ACM International Conference on Data Mining (ICDM-06), IEEE Proceedings pp:420-427.

Kashyap V. (1999). Design and Creation of Ontologies for Environmental Information Retrieval. Twelfth Workshop on Knowledge Acquisition, Modelling and Management Voyager Inn.

Khan L., Luo F. (2002). Ontology Construction for Information Selection. Proceedings of the 14th International IEEE Conference.

Kietz JU, Maedche A., Volz R. (2000). A Method for Semi-Automatic Ontology Acquisition from a Corporate Intranet. In : Aussenac-Gilles N, Biébow B. Szulman S (eds) EKAW'00 Workshop Proceedings 51 :4.1-4.14. http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-51/.

Maedche A., Staab S. (2000a). Mining Ontologies from text. Prceedings of the 12th International conference on Knowledge Engineering and Knowledge Management, Springer Lecture Notes in Artificial Intelligence.

Maedche A., Staab S. (2000b). Semi-automatic enginering of ontologies from text. Proceedings of the 12th International Conference on Software Engineering and Knowledge Engineering (SEKE'00) pp 231-239.

Maedche A., Staab S. (2001). Ontology Learning for the Semantic Web. IEEE Intelligent Systems, Special Issues on the Semantic Web, 16(2) : 72-79.

Michalsky R. (1980). Knowledge Acquisition through conceptual clustering: a theoretical framework and algorithm for partitioning data into conjonctive concepts. International Journal of Policy Analysis and Information Systems 4(3):219-243.

Moldovan D.I., Girju R. (2000). Domain-specific Knowledge Acquisition and Classification using Wordnet". Proceedings of FLAIRS2000 Conference.

Morik K., Kietz JU. (1989). A Bootstrapping Approach to Conceptual Clustering. In: Segre AM (ed) Proceedings of the Sixth International Workshop on Machine Learning. Morgan Kaufmann Publishers pp503-504.

Morin E. (1999). Des patrons lexico-syntaxiques pour aider au dépouillement terminologique. Traitement Automatique des Langues, vol 40(1), pp 143-166.

Navigli R., Velardi P. (2004). Learning Domain Ontologies from Document Warehouses and dedicated Web Sites. Computational Linguistics, MIT Press.

Nobécourt J. (2000). A method to build formal ontologies from text. In EKAW-2000 Workshop on ontologies and text.

Rubin D.L., Hewett M., Oliver D.E., Klein T.E. et Altman R.B. (2002). Automatic data acquisition into ontologies from pharmacogenetics relational data sources using declarative object definitions and XML. In: Proceedings of the Pacific Symposium on Biology.

Sanchez D., Moreno A. (2004). Creating ontologies from web documents. Proceedings of the Setè Congrés Catatà d'Intelligence Artificielle, IOS Press 113.

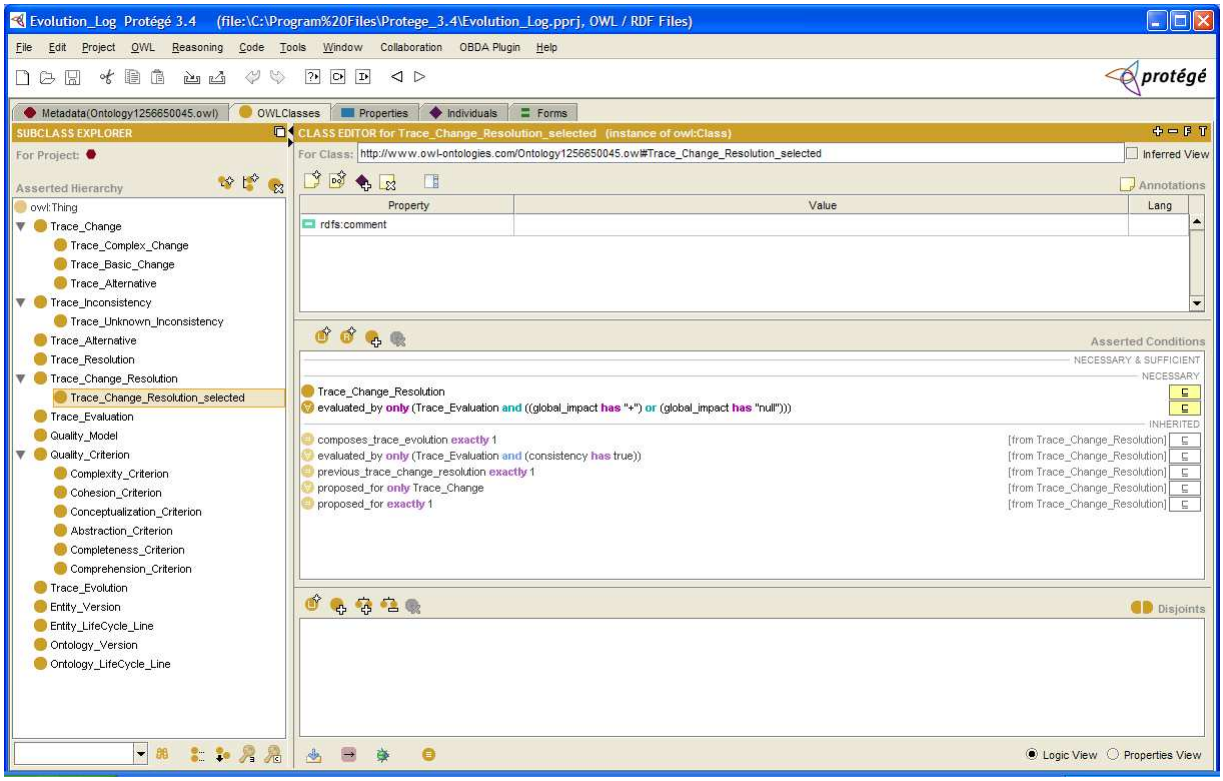
Sekiuchi R., Aoki C., Kurematsu M., Yamaguchi T. (1998). Doddle : A domain Ontology Rapid Development Environment. PRICAI.

Skrikant R., Agrawal R. (1995). Mining generelized association rules. In Proceedings of VLDB 95, pp 407-419.

Suryanto H., Compton P. (2001). Discovery of Ontologies from Knowledge Bases. Proceedings of the First International Conference on Knowledge Capture, The Association for Computing Machinery, pp171-178.

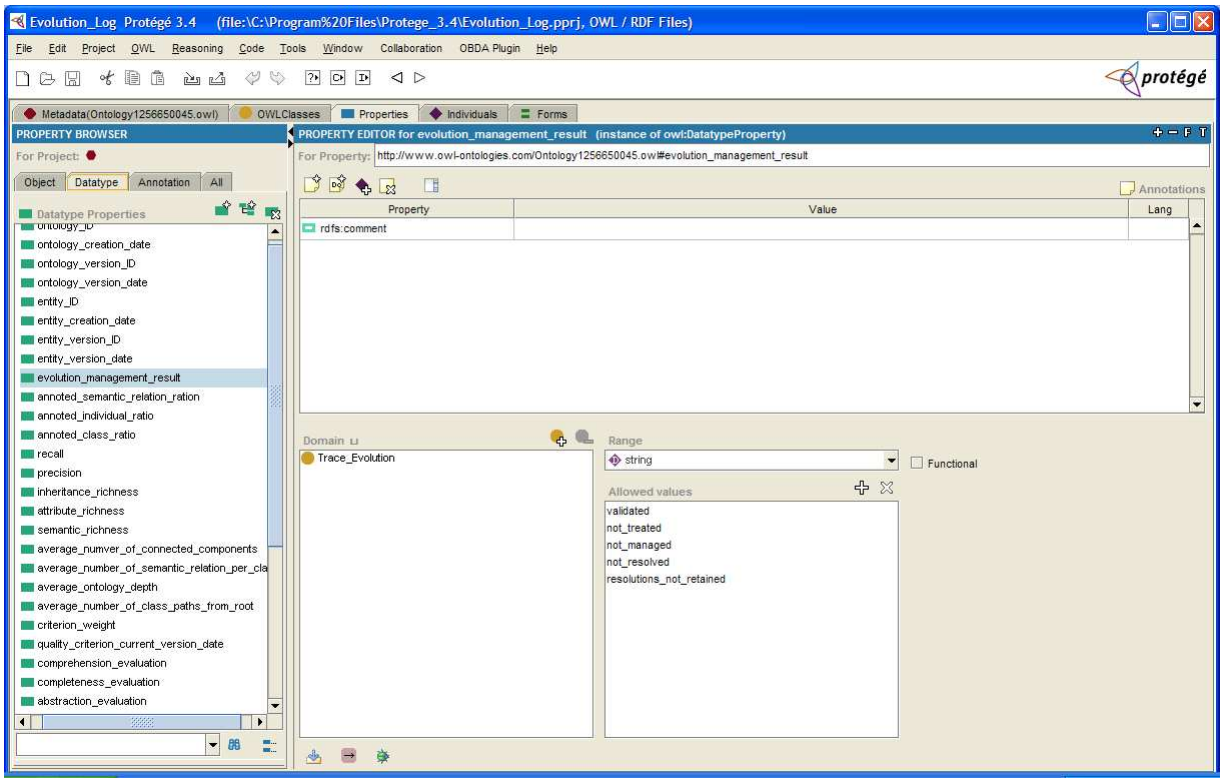
Annexe B : Ontologie Journal d'évolution* en OWL DL

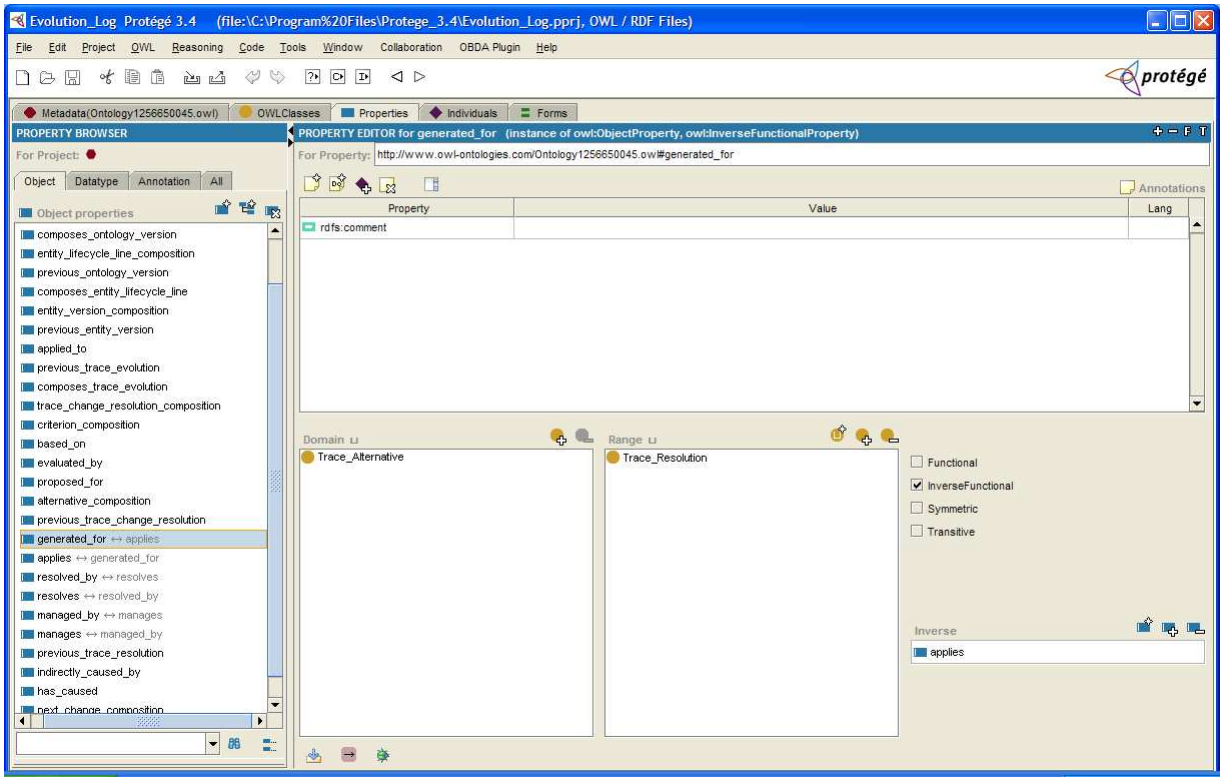
I- LISTE DES CLASSES



* Pour un objectif de partage, les primitives de l'ontologie Journal d'évolution ont été exprimées en anglais.

II- APERÇU DE LA LISTE DES PROPRIETES





Annexe C : Description des patrons CMP (complément)

I- PATRONS DE CHANGEMENTS BASIQUES

I.1- Patron de changements basiques Etendre la définition d'une classe

C'est une modification particulière dans le sens où la définition de la classe est enrichie sans supprimer les axiomes qui la définissent déjà. Cette opération correspond à un changement basique et s'approche de l'ajout d'une classe sauf qu'elle tient compte de la définition existante et vérifie un ensemble de contraintes supplémentaires.

Type des entités concernées	Arguments	Contraintes	Axiomes OWL DL (Syntaxe abstraite)*
Définir une collection d'union de classes			

- Une classe
- Une collection de classes
	- IdClasse
	- Id(s) Collection
	- Operateur (Union)
		La classe n'est disjointe (explicitement) d'aucune des classes constituant la collection.	`axiom ::= 'Class(' classID description1 ')'`
			Condition nécessaire et suffisante :
			`description1 ::= 'unionOf(' {description2} ')'`
			`Description2 ::= classID`

Définir une collection d'intersection de classes			

- Une classe
- Une collection de classes
	- IdClasse
	- Id(s) Collection
	- Operateur (Intersection)
		- Les classes constituant la collection ne sont pas toutes disjointes†.
		- La classe n'est disjointe d'aucune des classes constituant la collection.
			`axiom ::= 'Class(' classID description1 ')'`
			Condition nécessaire et suffisante :
			`description1 ::= 'intersectionOf('`
			`{description2} ')'`
			`Description2 ::= classID`

Définir une collection de complément de classes			

- Une classe
- Une collection de classes
	- IdClasse,
	- Id(s) Collection
	- Operateur (Complement)
		La classe n'est équivalente à aucune des classes constituant la collection.	`axiom ::= 'Class(' classID description1 ')'`
			Condition nécessaire et suffisante :
			`description1 ::= 'complementOf ('`
			`{description2} ')'`
			`Description2 ::= classID`

* http://www.w3.org/TR/owl-semantics/syntax.html

† Autrement l'intersection sera vide. Cette contrainte permet de notifier le cas où le changement crée une classe vide.

Définir une énumération			

- Une classe
- Une collection d'individus
	- IdClasse,
	- Id(s) Individus
		- La classe n'est pas disjointe des individus de la collection
		- La classe (si elle existe) a des instances appartenant uniquement aux individus spécifiés par la collection.
			`axiom ::= 'Class(' classID description1 ')'`
			Condition nécessaire et suffisante :
			`description1 ::= 'oneOf('{individualID} ')'`

Table 1. Synthèse du patron de changements basiques Etendre la définition d'une classe.

I.2- Patron de changements basiques Etendre l'instanciation d'un individu

Pour le peuplement de l'ontologie, nous distinguons une opération d'ajout d'un nouvel individu à l'ontologie, d'une opération d'extension de l'instanciation d'un individu – existant dans l'ontologie – à une autre classe.

L'ajout d'un individu peut correspondre à un peuplement sans instanciation c'est-à-dire juste un ajout d'un individu qui sera donc par défaut instance de la classe `owl:Thing` ou un peuplement avec instanciation en assignant l'individu à une classe, ou à un domaine ou co-domaine d'une propriété.

La définition de l'individu se trouve enrichie sans supprimer les instanciations qui le définissent déjà. Le tableau ci-après résume les propriétés de description d'un patron de changements basiques Etendre l'instanciation d'un individu à travers ses différentes spécialisations.

Type des entités concernées	**Arguments**	**Contraintes**	**Axiomes OWL DL (Syntaxe abstraite)**
Etendre l'instanciation d'un individu à une classe			

- Un individu
- Une classe à instancier
	- IdIndividu
	- IdClasse
		La classe n'est disjointe d'aucune des classes déjà instanciée par l'individu.	`Fact ::= individual`
			`individual ::= 'Individual(' individualID {'type(' type ')'}')'`
			
			`type ::= description1`
			`description1 ::=`
			`classID`
			`|'unionOf(' {description2}')'`
			`|'intersectionOf(' {description2 }')'`
			`|'complementOf(' {description2}')'`
			`Description2 ::= classID`

Etendre l'instanciation d'un individu à un domaine ou co-domaine de propriété			

- Un individu
- Une propriété
	- IdIndividu
	- IdPropriété
			`Fact ::= individual`
			`individual ::= 'Individual(' {value} ')'`
			
			`value ::=`
			`'value(' individualvaluedPropertyID individualID ')'`
			`|value(' datavaluedPropertyID dataLiteral ')'`

Table 2. Synthèse du patron de changements basiques Etendre l'instanciation d'un individu.

I.3- Liste des patrons de changements basiques

Patrons de changements basiques	
Classe de patrons	**Liste des patrons**
Classe	Ajouter (classe, union de classes, intersection de classes, complément de classes, énumération), Etendre définition (par une union de classes, une intersection de classes, un complément de classes, une énumération), Supprimer
Sous-classe	Ajouter, Supprimer
Equivalence de classe	Ajouter (à une classe, une union de classes, une intersection de classes, un complément de classes, une énumération), Supprimer
Disjonction de classe	Ajouter (à une classe, une union de classes, une intersection de classes, un complément de classes, une énumération), Supprimer
Restriction de valeur	Ajouter (égalité, universelle, existentielle), Supprimer
Restriction de cardinalité	Ajouter (maximale, minimale, exacte), Supprimer
Propriété	Ajouter, Supprimer
Domaine de propriété	Ajouter (classe, union de classes, intersection de classes, complément de classes, énumération, restriction de valeur, restriction de cardinalité), Modifier, Supprimer
Co-domaine de propriété	Ajouter (propriété objet (union de classes, intersection de classes), propriété de données), Modifier (propriété objet, propriété de données), Supprimer (propriété objet, propriété de données)
Sous-Propriété	Ajouter, Supprimer
Equivalence de propriété	Ajouter, Supprimer
Propriété fonctionnelle	Mettre (propriété objet), Annuler
Propriété inverse fonctionnelle	Mettre, Annuler
Propriété inverse	Ajouter, Supprimer
Propriété symétrique	Mettre, Annuler
Propriété transitive	Mettre, Annuler
Individu	Ajouter, Etendre instanciation (classe, domaine ou co-domaine de propriété), Supprimer, Supprimer instanciation (classe, domaine ou co-domaine de propriété)
Différence d'individus	Ajouter, Supprimer
Identité d'individus	Ajouter, Supprimer

Table 3. Synthèse de la liste des patrons de changements basiques définis.

II- PATRONS DE CHANGEMENTS COMPLEXES

II.1- Patron de changements complexes d'ajout d'une relation de sous-classe en multi-héritage

Patron : Ajouter une relation sous-classe en multi-héritage		
Arguments	Objet du changement	- ID de la sous-classe (sub_classID)
	Entités référencées	- ID de la sous-classe (sub_classID) -Id(s) des superclasses de la sous-classe {super_classID}
	Nombre superclasses	- Nb
Contraintes		La sous-classe n'est disjointe à aucune des superclasses : ⇨ Pour chaque super_classID de {super_classID} :

	¬ (sub_classID `disjointWith` super_classID). `Les superclasses et leurs ascendants ne sont pas disjoints entre eux :` ⇨ Pour chaque super_classID de {super_classID} : ¬ (super_classID `disjointWith` {super_classID}). ¬ (`Ascendants` (super_classID) `disjointWith` `Ascendants` {super_classID}).
Séquence	Nb fois : PChB_AjoutSous-Classe_1 (sub_classID, super_classID)

Table 4. Synthèse de la description du patron de changements complexes d'ajout d'une relation de sous-classes en multi-héritage.

II.2- Liste des Patrons de changements complexes

Liste des Patrons de changements complexes	
Sous-hiérarchie	Ajouter une classe et ses sous-classes.
Multi-héritage	Ajouter une sous-classe en multi-héritage.
Fusion	Fusionner deux classes.
Généralisation	Remonter une classe.
Spécialisation	Descendre une classe.
Suppression	Supprimer une classe en rattachant ses propriétés à ses sous-classes, et ses instances et sous-classes à une superclasse.
Domaine de propriété	Etendre le domaine d'une propriété à une superclasse, Restreindre le domaine d'une propriété à une sous-classe.
Co-domaine de propriété	Etendre le co-domaine d'une propriété à une superclasse, Restreindre le co-domaine d'une propriété à une sous-classe.
Instanciation	Etendre l'instanciation d'un individu multiple (ajouter plusieurs instanciations à un individu existant).
Agréger propriétés	Agréger propriétés de deux classes.
Restriction de valeur complexe (union, intersection ou complément de classes)	Restriction de valeur universelle complexe, Restriction de valeur existentielle complexe, Restriction de valeur d'égalité complexe.

Table 5. Synthèse de la liste des patrons de changements complexes définis.

III- PATRONS D'INCOHERENCES

Liste des patrons d'incohérences

Patrons d'incohérences	
Disjonction	Subsomption, Instanciation, Ascendants
Restriction de valeur	Restriction universelle inapplicable, Incompatibilité de restrictions universelle et existentielle
Suppression d'une classe	Dépendances obsolètes (propriétés, sous-classes et instances)
Définition complexe de domaine ou co-domaine	Intersection de domaines disjoints, Intersection de co-domaines disjoints

Table 6. Synthèse de la liste des patrons d'incohérences définis.

IV- PATRONS D'ALTERNATIVES

Liste des patrons d'alternatives

Patrons d'alternatives	
Résolution d'incohérences de disjonction liée à la subsomption par classe hybride	Définir une classe hybride pour une subsomption, Relier toutes les subsomptions à une classe hybride.
Résolution d'incohérences de disjonction liée à l'instanciation par classe hybride	Définir une classe hybride pour une instance, Relier toutes les instances à une classe hybride.
Résolution d'incohérences de disjonction basée sur les définitions de classe	Etendre définition de toutes les subsomptions, Etendre définition d'une sous-classe, Etendre définition d'une sous-classe par rapport aux ascendants disjoints.
Résolution d'incohérences de subsomption par généralisation	Définir une subsomption généralisée.
Résolution d'incohérences de restriction de valeur	Résolution de restriction universelle inapplicable, Résolution d'incompatibilité de restrictions universelle-existentielle, Résolution d'incompatibilité de restrictions existentielle-universelle.
Résolution d'incohérences de dépendances obsolètes	Détacher les dépendances obsolètes, Reclasser et Redistribuer les dépendances obsolètes
Résolution d'intersection de domaines ou co-domaines disjoints	Transformer domaine intersection en domaine union, Transformer co-domaine intersection en co-domaine union

Table 7. Synthèse de la liste des patrons d'alternatives définis.

Annexe D : Description des patrons CMP en UML (complément)

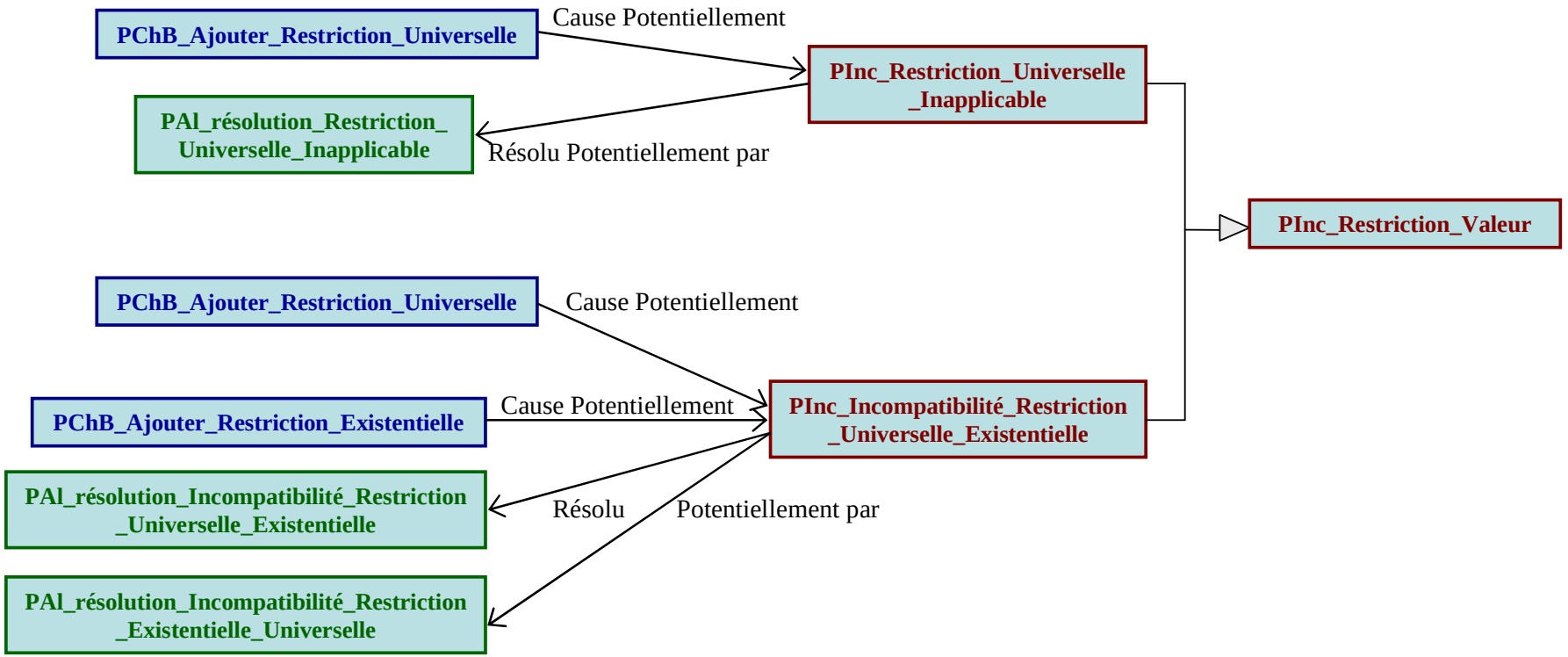


Figure 1. Représentation graphique en UML de la modélisation d'un exemple de patron de d'incohérence de restriction de valeur avec ses spécialisations, les patrons de changements qui leurs sont liés et les patrons d'alternatives qui peuvent les résoudre.

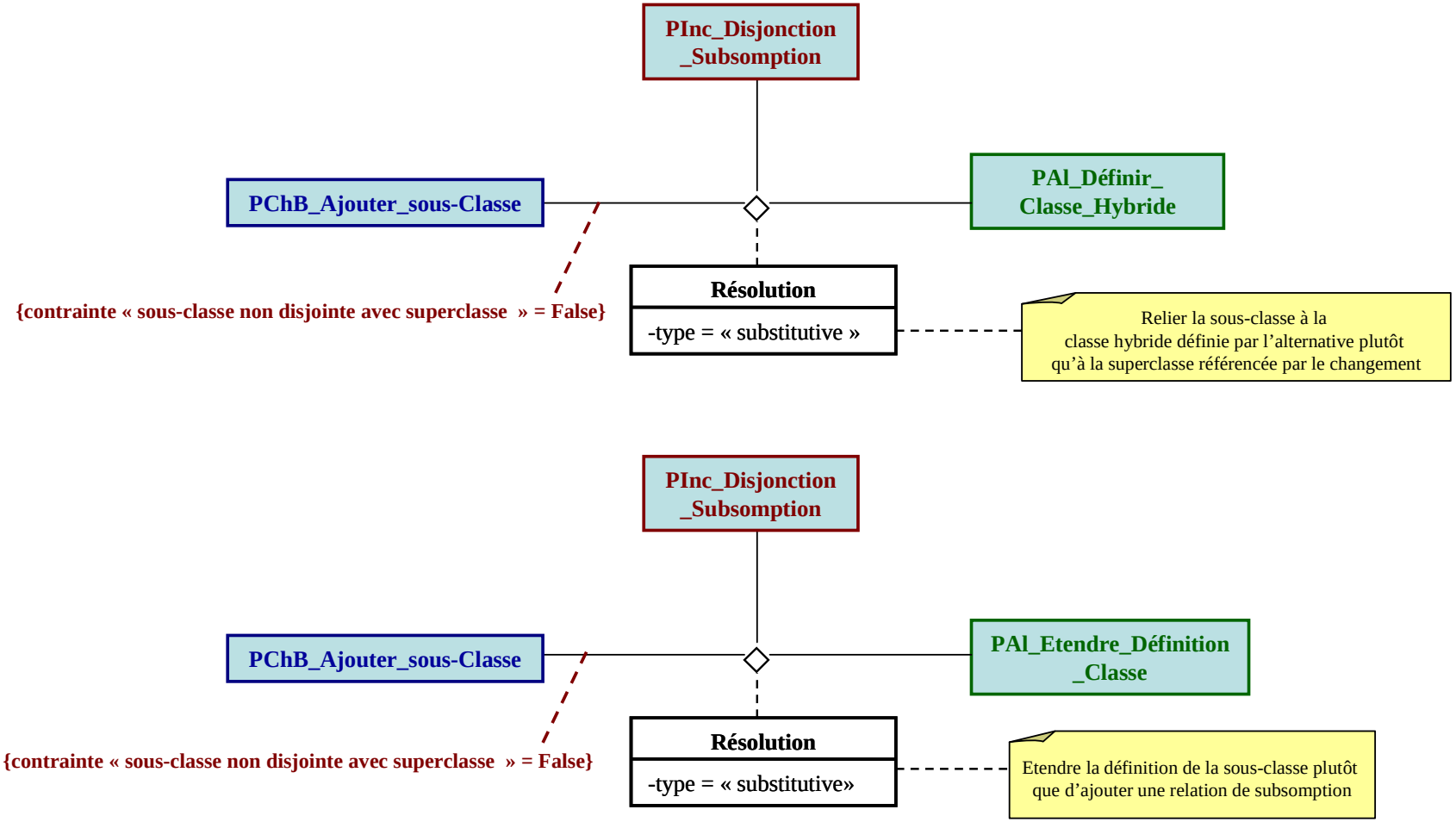


Figure 2. Représentation graphique en UML de la modélisation des résolutions du patron de changement « Ajouter sous-classe » pour le patron d'incohérences qu'il peut causer : « Disjonction de subsomption ».

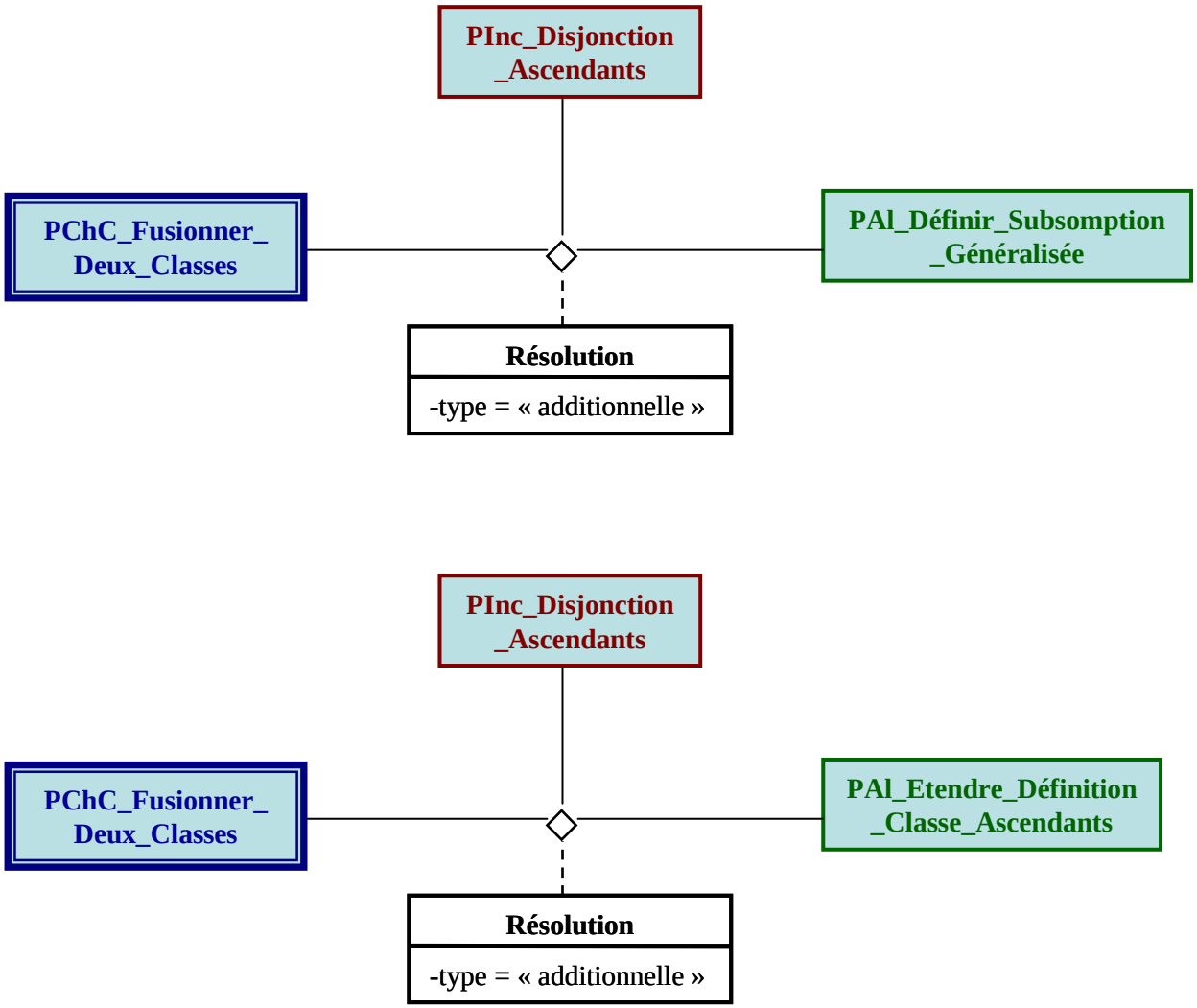
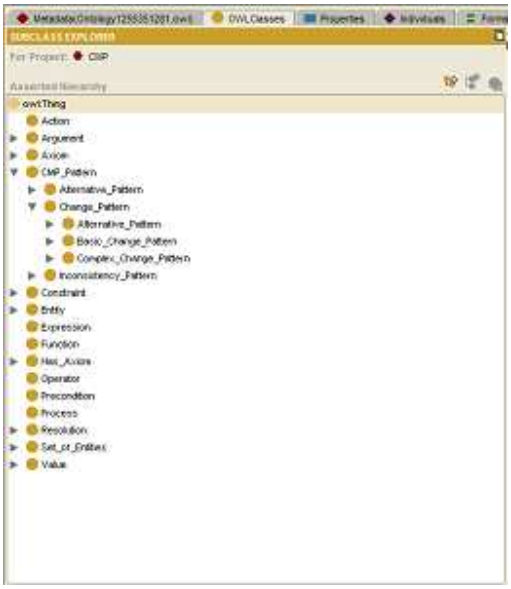
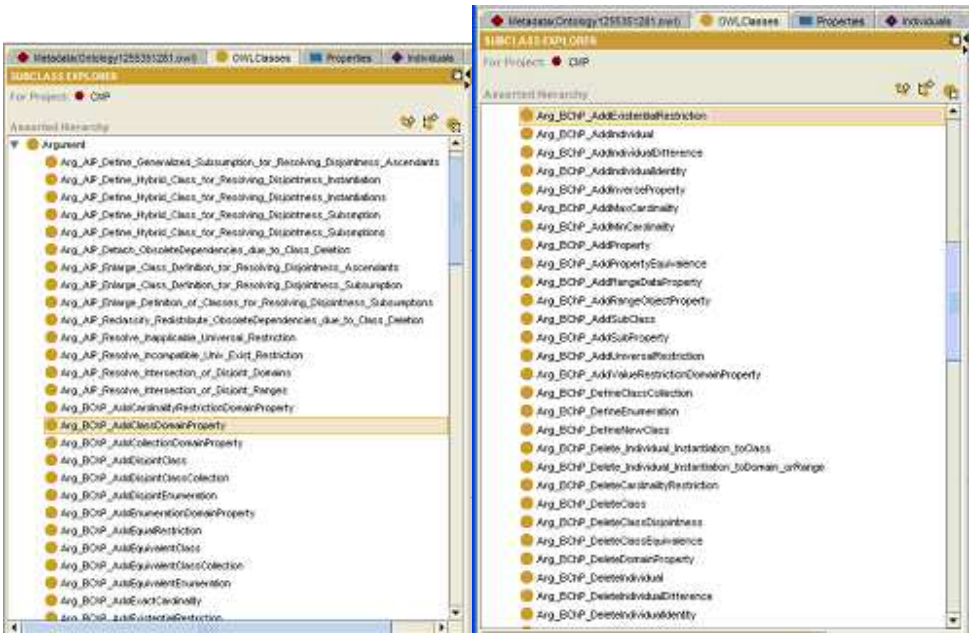


Figure 3. Représentation graphique en UML de la modélisation des résolutions du patron de changement « Fusionner deux classes » pour le patron d'incohérences qu'il peut causer : « Disjonction d'ascendants »

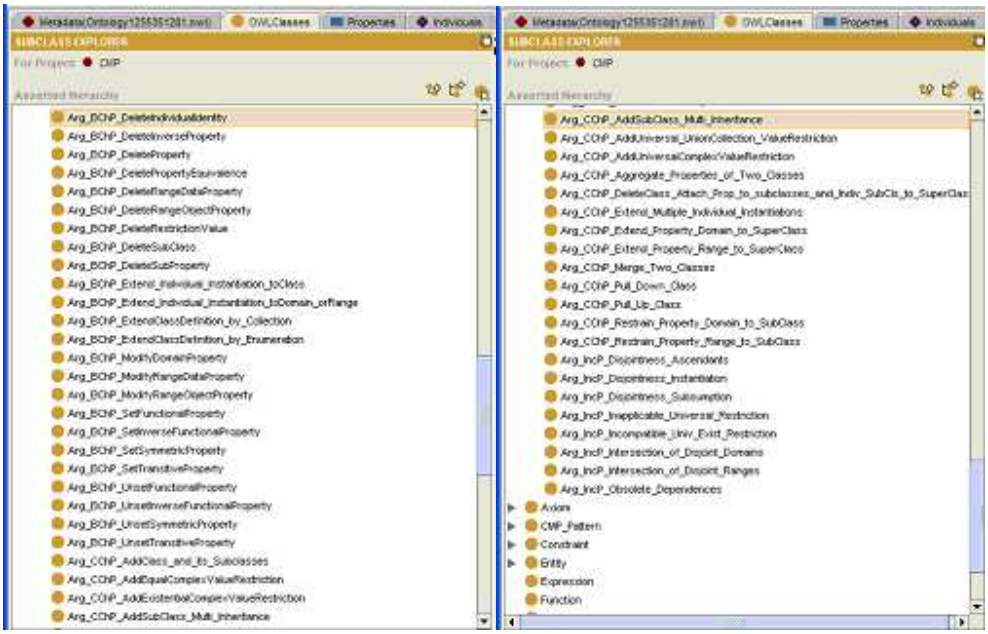
Annexe E : Ontologie CMP* en OWL DL



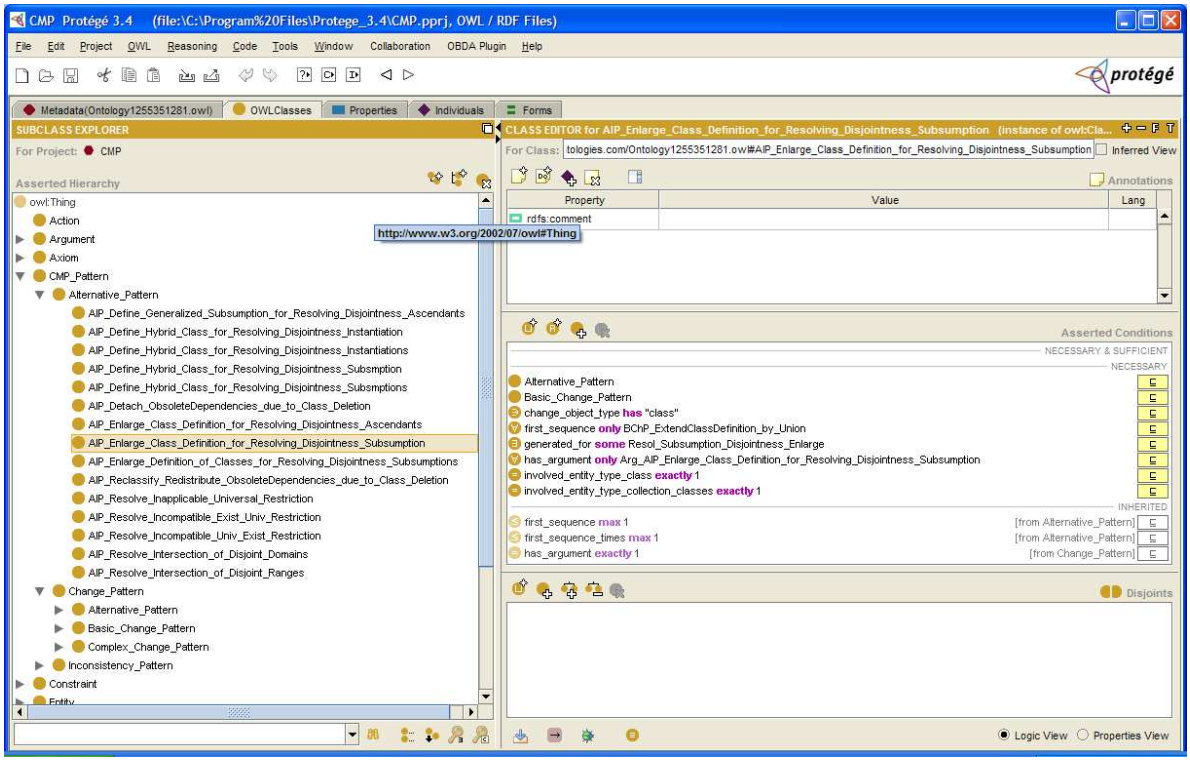
I- LISTE DES CLASSES ARGUMENT



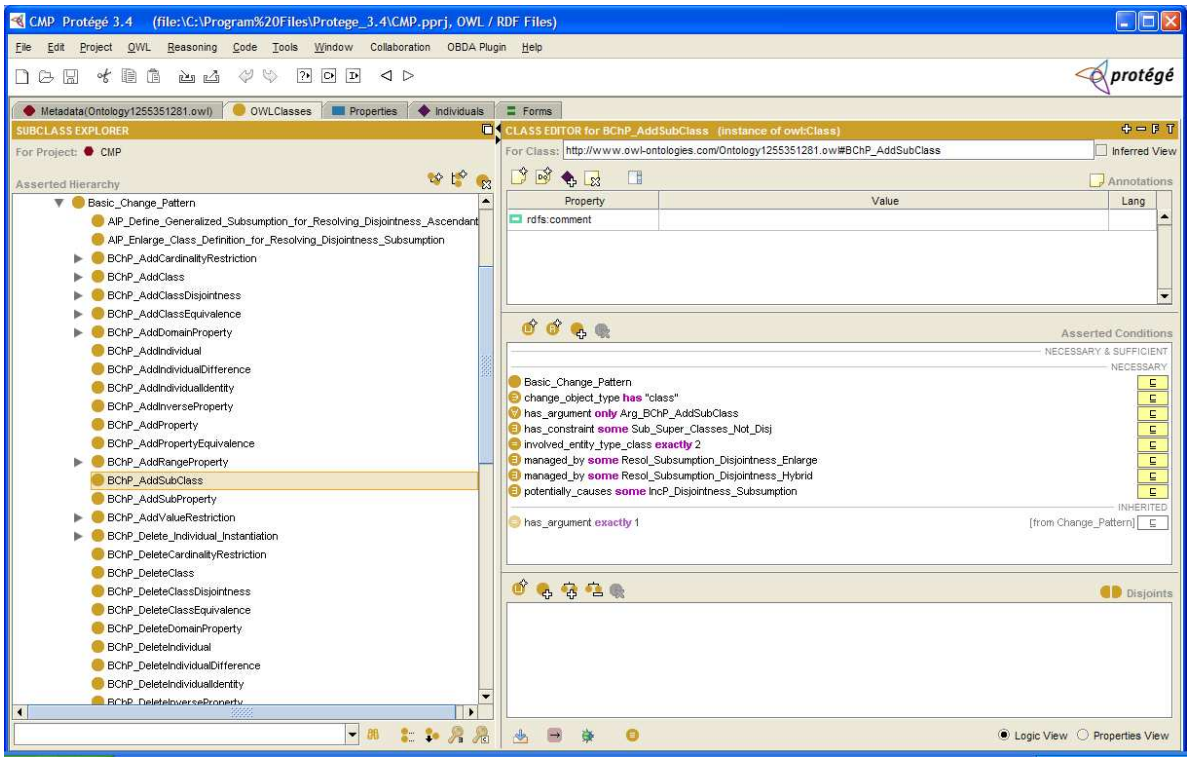
* Pour un objectif de partage, les primitives de l'ontologie CMP ont été exprimées en anglais.

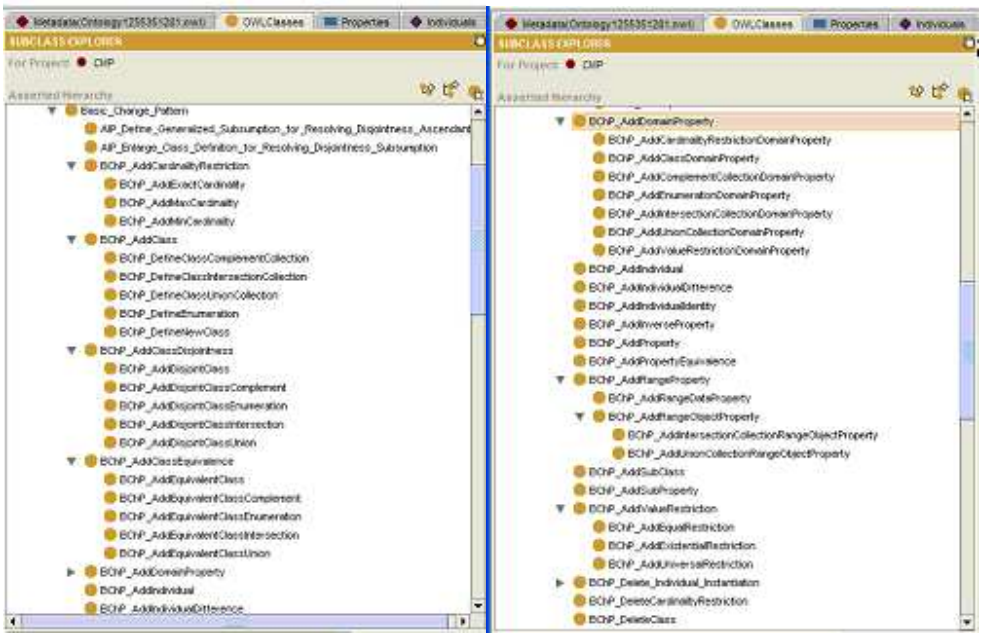


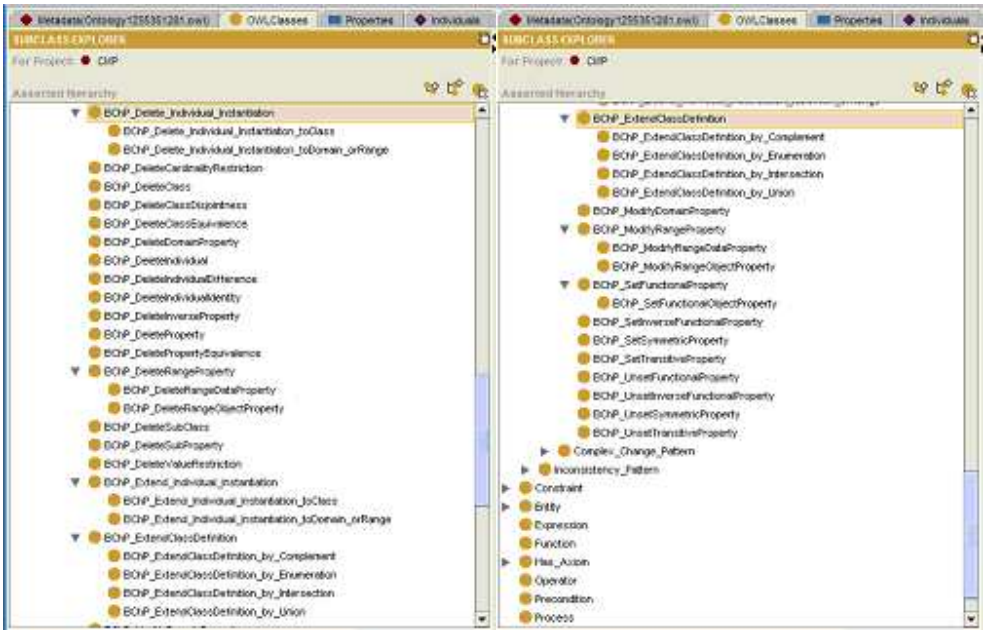
II- LISTE DES CLASSES PATRONS D'ALTERNATIVES



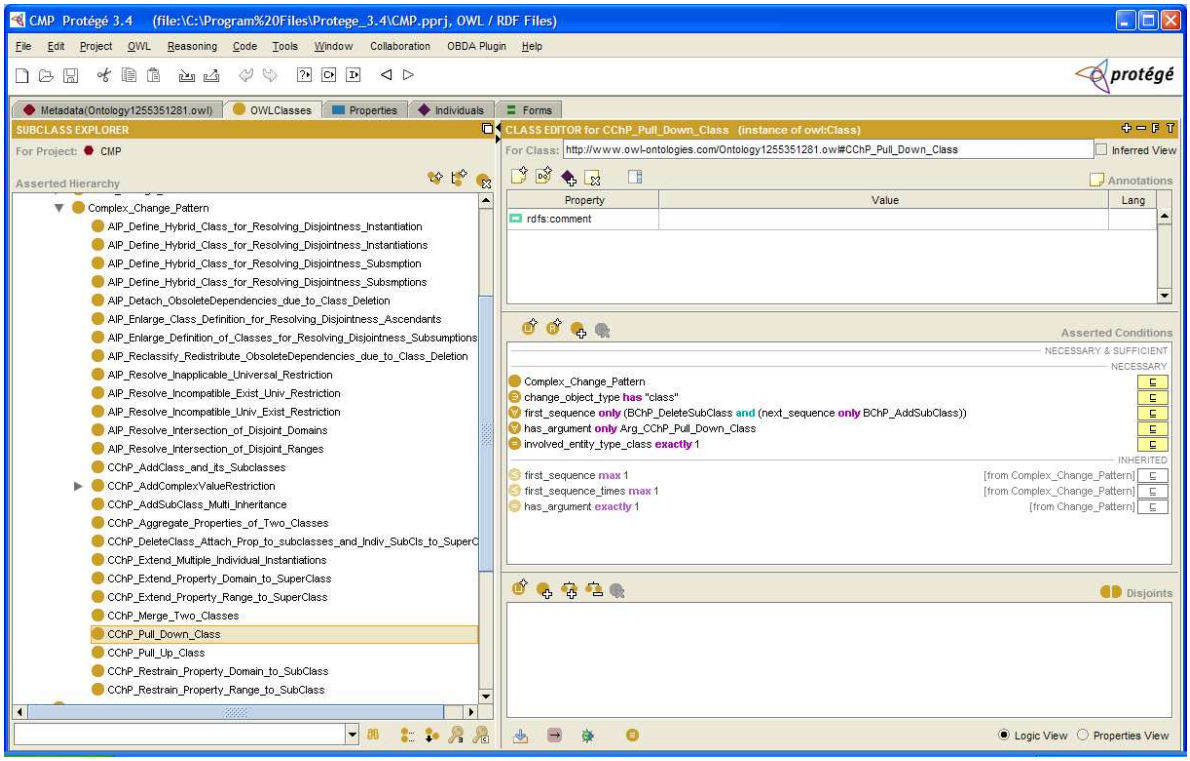
III- LISTE DES CLASSES PATRONS DE CHANGEMENTS BASIQUES



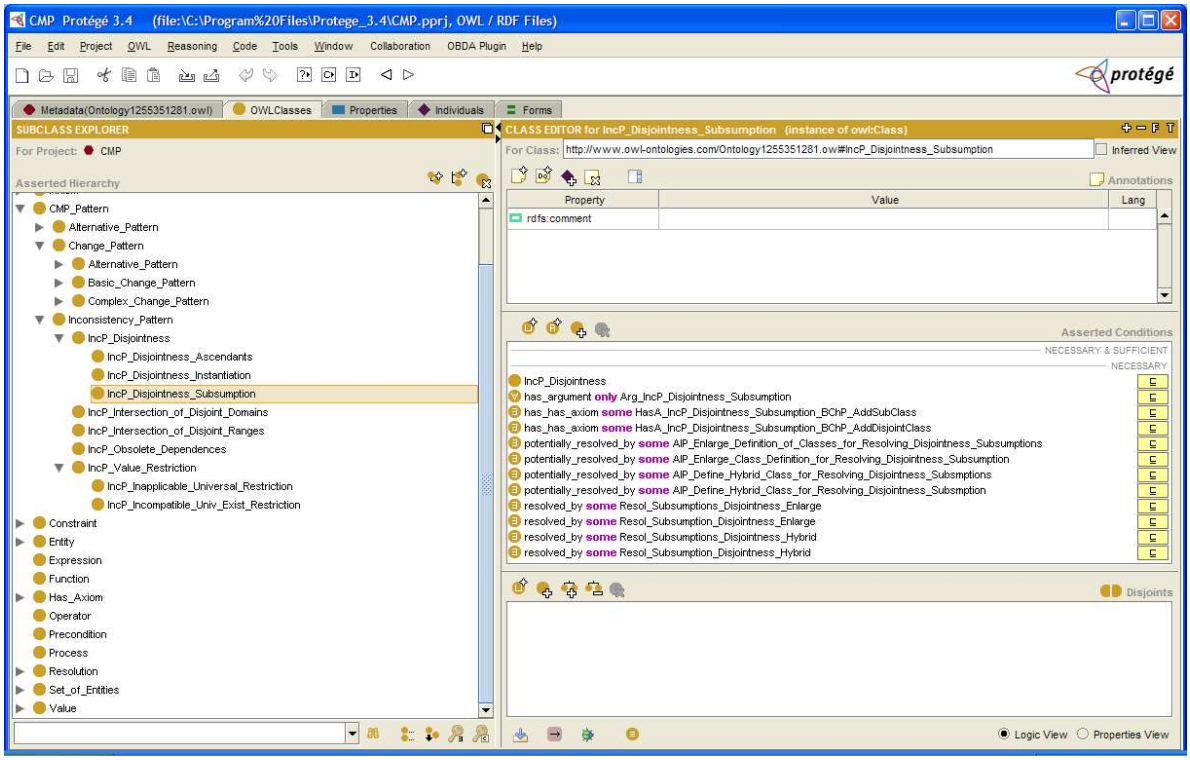




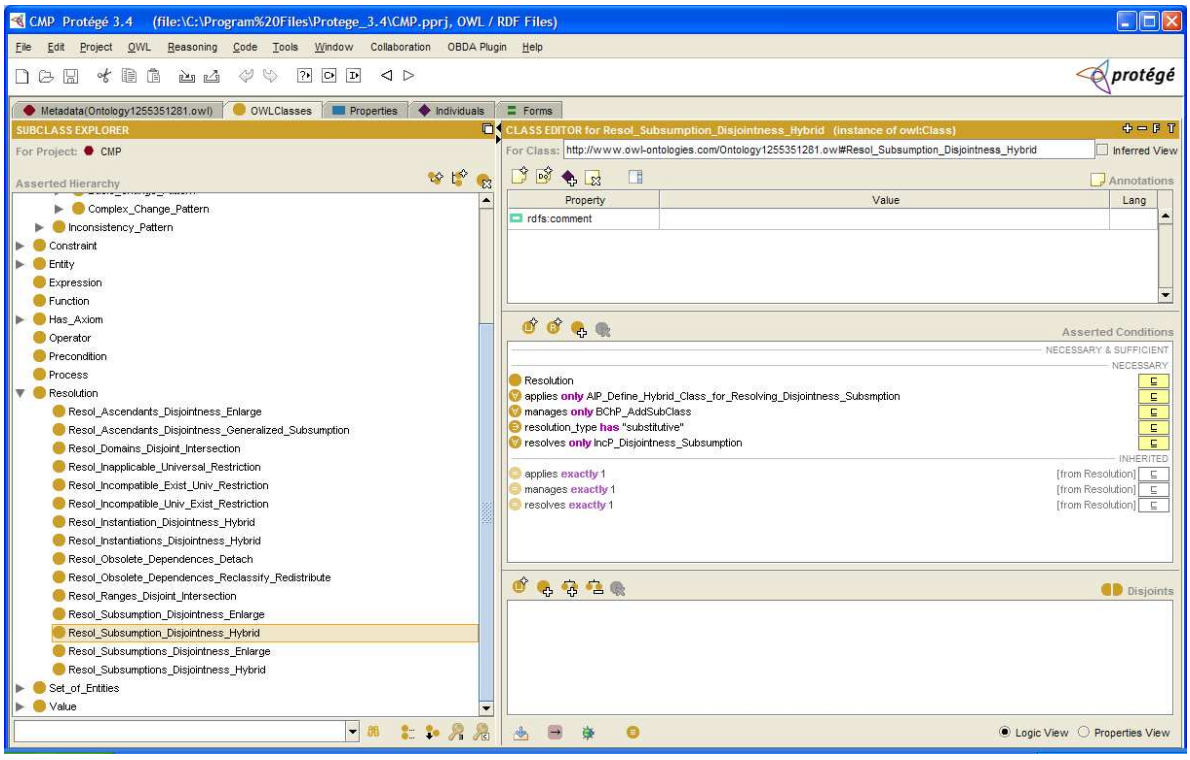
IV- LISTE DES CLASSES PATRON DE CHANGEMENTS COMPLEXES



V.- LISTE DES CLASSES PATRON D'INCOHERENCES



VI- LISTE DES CLASSES RESOLUTION



Annexe F : Techniques d'alignement d'ontologie

I- ALIGNEMENT DE SCHEMAS D'ONTOLOGIE

L'accroissement du nombre d'ontologies spécifiant un même domaine, les recherches en ingénierie ontologique se sont portées sur la comparaison et l'alignement d'ontologies en vue d'établir des correspondances, de tracer l'évolution d'une ontologie à travers la comparaison des différentes versions, de créer un ensemble d'axiomes ou de règles de rapprochement entre ontologies, de générer des médiateurs de requêtes, ou encore afin de constituer à partir d'une base de connaissances source, une base de connaissance cible qui servira de référence pour le domaine[*].

Trois principales notions sont à distinguer (Bouquet et al., 2005):

- Le *mapping* correspondant à une expression formelle (relation orientée avec des propriétés mathématiques) qui établit des relations sémantiques (équivalence, subsomption, etc.) appelées encore des règles de correspondance, entre les concepts et les relations similaires de différentes ontologies ;
- La mise en correspondance d'ontologies ou de parties d'ontologies appelée *matching*, permet de faire ressortir des appariements de concepts et de relations pour définir une carte sémantique entre deux ontologies.
- L'alignement d'ontologies désignant aussi bien le processus d'alignement que le produit résultant. Le processus établit la cohérence de plusieurs ontologies à travers un consensus mutuel en appliquant des opérations de matching pour ressortir les correspondances entre ontologies exprimées sous forme de mappings.

I.1- Classification des techniques d'alignement de schémas d'ontologie

Les méthodes d'alignement combinent différentes techniques en fonction de leurs objectifs particuliers. En tenant compte de la classification des techniques d'alignement de schémas selon le type des sources en entrée (sens descendant) et le type des techniques elles-mêmes (sens ascendant) (Shvaiko & Euzenat, 2005) (Euzenat & Shvaiko, 2007a), nous nous focalisons sur les techniques d'alignement locales regroupant des méthodes basiques de mesure de similarités ontologiques entre entités (niveau élément), et les techniques globales basées sur l'agrégation des résultats des méthodes locales pour établir un alignement global considérant l'ontologie dans sa totalité (niveau structure), correspondant à des techniques structurelles et sémantiques (figure 1).

I.2- Techniques structurelles

Les techniques structurelles sont basées sur les relations entre les entités des ontologies, les attributs décrivant les entités et les contraintes appliquées sur leur valeur (Euzenat & Shvaiko, 2007b) (Kalfoglou & Schorlemmer, 2003). Elles emploient des techniques de comparaison de types et de graphes.

[*] http://www.ontologymatching.org/

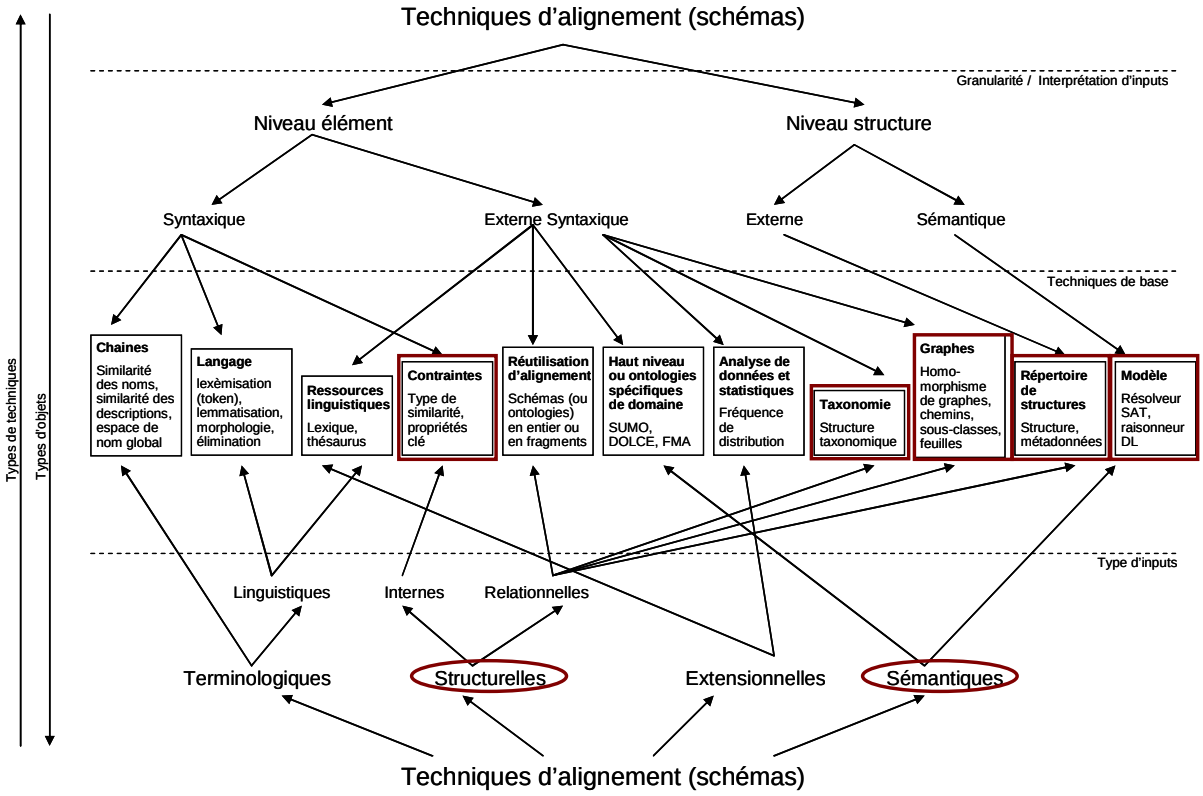


Figure 1. Classification des techniques d'alignement de schémas d'ontologie.

I.3- Techniques sémantiques

Les techniques sémantiques se basent sur la sémantique des ontologies en appliquant des démonstrateurs de preuves et en s'appuyant sur des sources externes de connaissances formelles (Euzenat & Shvaiko, 2007b) (Kalfoglou & Schorlemmer, 2003).

I.4- Techniques niveau élément et niveau structure

Certaines de ces méthodes ont été implémentées à travers des outils d'alignement. Un état de l'art complet sur les méthodes et les systèmes d'alignement d'ontologies a été présenté dans (Euzenat et al., 2004), (Le Bach, 2006) et (Mochol, 2009).

Voici quelques exemples de techniques d'alignement appliqués niveau élément et niveau structure, avec les principes inhérents et les systèmes qui les appliquent (table 1).

Techniques d'alignement niveau élément	
Techniques d'alignement basées sur les contraintes	
▪ Comparaison de propriétés datatype Integer < réel {a,c,g,t} [1-10] < {a,c,g,u,t}+ ▪ multiplicity Comparaison [1 - 1] < [1 - 10] Le résultat de comparaison peut être ramené en une	**OLA** (INRIA Rhône-Alpes et Université de Montréal). **COMA** (Université de Leipzig). **ASCO2** (Le Bach, 2006).

distance en estimant le ratio de couverture de domaine de chaque propriété datatype.	
Techniques d'alignement niveau structure	
Techniques d'alignement basées sur les arbres	
▪ fils Deux éléments non feuilles de la hiérarchie sont structurellement similaires si leur ensembles de fils directs sont hautement similaires. ▪ feuilles Deux éléments non feuilles de la hiérarchie sont structurellement similaires si leur ensembles de feuilles sont hautement similaires et ce, quelque soit la similarité entre leur fils directs.	**Cupid** (Université de Washington, Microsoft Corporation et Université de Leipzig). **COMA** (Université de Leipzig).
Techniques d'alignement basées sur les graphes	
▪ calcul itératif par rapport à un point fixe (iterative fix point computation) Si les voisins de deux nœuds pris de chacun des graphes des ontologies à aligner sont similaires, alors ces deux nœuds le sont aussi.	**SF** **OLA** (INRIA Rhône-Alpes et Université de Montréal). **ASCO 1,2,3** (Le Bach, 2006).
Techniques d'alignement basées sur les modèles	
▪ Satisfiabilité propositionnelle (SAT) ou les logiques de description Le principe des techniques basées sur les DL est qu'un matching, exprimé en axiomes, implique une relation entre les contextes des deux ontologies : Axioms → rel ($context_1$, $context_2$) ($Cls_1 \leftrightarrow Cls_2$) $\wedge$ ($sousCls_1 \leftrightarrow sousCls_2$) → ($Cls_1 \wedge sousCls_1$) $\leftrightarrow$ ($Cls_2 \wedge sousCls_2$) tels que ($Cls_1 \wedge sousCls_1$) et ($Cls_2 \wedge sousCls_2$) sont respectivement $context_1$ et $context_2$.	**CtxMatch** (Université de Trento). **S-Match** (Université de Trento). **ASCO 2** (Le Bach, 2006).

Table 1. Exemples de techniques d'alignement.

REFERENCES

Bouquet, P., Ehrig, M., Euzenat, E., Franconi, E., Hitzler, P., Krötzsch, M. et al. (2005) Specification of a common framework for characterizing alignment. Délivrable D2.2.1 du projet KWEB EU-IST-2004-507482, version 2.0. http://knowledgeweb.semanticweb.org/semanticportal/deliverables/D2.2.1v2.pdf

Euzenat, J., Le Bach, T., Barrasa, J., Bouquet, P., De Bo, J., Dieng, R., et al. (2004) State of the art on current alignment techniques. Délivrable D2.2.3 du projet KWEB EU-IST-2004-507482.

Euzenat, J., Shvaiko, P. (2007a). Support de Cours Ontology Alignement, Ecole d'été Summer School Semantic Web SSSW'07, organisée dans le cadre du projet KnowledgeWeb (http://knowledgeweb.semanticweb.org/semanticportal/sewView/frames.html).

Euzenat, J., Shvaiko, P. (2007b). Ontology Matching. Springer-Verlag, Heidelberg (DE). ISBN: 3-540-49611-4. Lien : http://book.ontologymatching.org/

Kalfoglou, Y. Schorlemmer, M. (2003) Ontology mapping: the state of the art. Knowledge Engineering Review, 18(1):1–31.

Le Bach, T. (2006) Construction d'un Web sémantique multi-points de vue. Thèse de doctorat. L'école des mines de Paris, Sophia Antipolis

Mochol, M. (2009). The Methodology for Finding Suitable Ontology Matching Approaches. These de doctorat. Institut d'informatique et mathématiques. Université libre de Berlin. http://www.diss.fu-berlin.de/diss/servlets/MCRFileNodeServlet/FUDISS_derivate_000000005051/thesis.pdf?hosts=

Shvaiko, P., Euzenat, J. (2005) A Survey of Schema-based Matching Approaches. Journal on Data Semantics, JoDS. Vol. IV. Lien: http://www.dit.unitn.it/~p2p/RelatedWork/Matching/JoDS-IV-2005_SurveyMatching-SE.pdf.

Annexe G : Exemple d'exécution des algorithmes du chapitre 5

I- TRACE D'EXECUTION DE L'ALGORITHME DE LOCALISATION DES INCOHERENCES (CHAPITRE 5, TABLE 6)

Afin d'illustrer l'exécution de l'algorithme de localisation, reprenons l'exemple de l'ontologie O illustrée dans le chapitre 4 et définie comme suit :

$\{\texttt{Animal} \sqsubseteq \texttt{Fauna-Flora},\ \texttt{Plant} \sqsubseteq \texttt{Fauna-Flora},\ \texttt{Carnivorous-Plant} \sqsubseteq \texttt{Plant},\ \texttt{Plant} \sqsubseteq \neg\texttt{Animal}\}$

Et le changement *Ch1* défini par l'axiome :

$\texttt{Carnivorous-Plant} \sqsubseteq \texttt{Animal}$

L'exécution de l'algorithme commence par un premier appel du raisonneur pour vérifier la cohérence de $\alpha_{Ch1} = \{\texttt{Carnivorous-Plant} \sqsubseteq \texttt{Animal}\}$.

$\Omega \leftarrow \{\texttt{Carnivorous-Plant} \sqsubseteq \texttt{Animal}\}$

Ensuite une première itération par rapport à Ω, est lancée :

Itération1

$\Omega' \leftarrow \phi$

$O' = \{\texttt{Carnivorous-Plant} \sqsubseteq \texttt{Animal}\}$

Ω'	$\beta_1 (\in O \setminus O')$	$\beta_2 (\in O')$
$\{\{\texttt{Carnivorous-Plant} \sqsubseteq \texttt{Animal}\} \cup \{\texttt{Animal} \sqsubseteq \texttt{Fauna-Flora}\}\}$	$\texttt{Animal} \sqsubseteq \texttt{Fauna-Flora}$	$\texttt{Carnivorous-Plant} \sqsubseteq \texttt{Animal}$
$\{\{\texttt{Carnivorous-Plant} \sqsubseteq \texttt{Animal}\} \cup \{\texttt{Animal} \sqsubseteq \texttt{Fauna-Flora}\}\}$	$\texttt{Plant} \sqsubseteq \texttt{Fauna-Flora}$	$\texttt{Carnivorous-Plant} \sqsubseteq \texttt{Animal}$
$\{\{\{\texttt{Carnivorous-Plant} \sqsubseteq \texttt{Animal}\} \cup \{\texttt{Animal} \sqsubseteq \texttt{Fauna-Flora}\}\} \cup \{\{\texttt{Carnivorous-Plant} \sqsubseteq \texttt{Animal}\} \cup \{\texttt{Carnivorous-Plant} \sqsubseteq \texttt{Plant}\}\}\}$	$\texttt{Carnivorous-Plant} \sqsubseteq \texttt{Plant}$	$\texttt{Carnivorous-Plant} \sqsubseteq \texttt{Animal}$
$\{\{\{\texttt{Carnivorous-Plant} \sqsubseteq \texttt{Animal}\} \cup \{\texttt{Animal} \sqsubseteq \texttt{Fauna-Flora}\}\} \cup \{\{\texttt{Carnivorous-Plant} \sqsubseteq \texttt{Animal}\} \cup \{\texttt{Carnivorous-Plant} \sqsubseteq \texttt{Plant}\}\} \cup \{\{\texttt{Carnivorous-Plant} \sqsubseteq \texttt{Animal}\} \cup \{\texttt{Plant} \sqsubseteq \neg\texttt{Animal}\}\}\}$	$\texttt{Plant} \sqsubseteq \neg\texttt{Animal}$	$\texttt{Carnivorous-Plant} \sqsubseteq \texttt{Animal}$

A la fin de l'itération nous avons $\Omega \leftarrow \Omega'$ ce qui donne :

$\Omega = \{$

$\Omega[1] : \{\{\texttt{Carnivorous-Plant} \sqsubseteq \texttt{Animal}\} \cup \{\texttt{Animal} \sqsubseteq \texttt{Fauna-Flora}\}$

Ω[2] : {{Carnivorous-Plant ⊑ Animal} ∪ {Carnivorous-Plant ⊑ Plant}} ∪
Ω[3] : {{Carnivorous-Plant ⊑ Animal} ∪ {Plant ⊑ ¬Animal}}
}
nb_candidate ← |Ω| donne nb_candidate = 3

Pour chacune de ces trois sous-ontologies candidates constituant Ω (Ω[1], Ω[2], Ω[3]), le raisonneur est appelé pour vérifier la cohérence. Aucune incohérence n'est détectée (cohérence = true).

A la 2ème itération par rapport à on a Ω :

Itération2

Ω' ← $\emptyset$

Pour *O'*= Ω[1] = {{Carnivorous-Plant ⊑ Animal} ∪ {Animal ⊑ Fauna-Flora}}

Ω'	β_1	β_2
{{{Carnivorous-Plant ⊑ Animal} ∪ {Animal ⊑ Fauna-Flora}} ∪ {Plant ⊑ Fauna-Flora}}	Plant ⊑ Fauna-Flora	Animal ⊑ Fauna-Flora
{**Ω' précédante** ∪ {{{Carnivorous-Plant ⊑ Animal} ∪ {Animal ⊑ Fauna-Flora}} ∪ {Carnivorous-Plant ⊑ Plant}}	Carnivorous-Plant ⊑ Plant	Carnivorous-Plant ⊑ Animal
{**Ω' précédante** ∪ {{{Carnivorous-Plant ⊑ Animal} ∪ {Animal ⊑ Fauna-Flora}} ∪ {Plant ⊑ ¬Animal}}	Plant ⊑ ¬Animal	Carnivorous-Plant ⊑ Animal

Pour *O'*= Ω[2] = {{Carnivorous-Plant ⊑ Animal} ∪ {Carnivorous-Plant ⊑ Plant}}

Ω'	β_1	β_2
{**Ω' précédante** ∪ {{{Carnivorous-Plant ⊑ Animal} ∪ {Carnivorous-Plant ⊑ Plant}} ∪ {Animal ⊑ Fauna-Flora}}	Animal ⊑ Fauna-Flora	Carnivorous-Plant ⊑ Animal
{**Ω' précédante** ∪ {{{Carnivorous-Plant ⊑ Animal} ∪ {Carnivorous-Plant ⊑ Plant}} ∪ {Plant ⊑ Fauna-Flora}}	Plant ⊑ Fauna-Flora	Carnivorous-Plant ⊑ Plant
{**Ω' précédante** ∪ {{{Carnivorous-Plant ⊑ Animal} ∪ {Carnivorous-Plant ⊑ Plant}} ∪ {Plant ⊑ ¬Animal}}	Plant ⊑ ¬Animal	Carnivorous-Plant ⊑ Animal

Pour *O'*= Ω[3] = {{Carnivorous-Plant ⊑ Animal} ∪ {Plant ⊑ ¬Animal}}

Ω'	β_1	β_2
{**Ω' précédante** ∪ {{{Carnivorous-Plant ⊑ Animal} ∪ {Plant ⊑ ¬Animal}} ∪ {Animal ⊑ Fauna-Flora}}	Animal ⊑ Fauna-Flora	Carnivorous-Plant ⊑ Animal
{**Ω' précédante** ∪ {{{Carnivorous-Plant ⊑ Animal} ∪ {Plant ⊑ ¬Animal}} ∪ {Plant	Plant ⊑ Fauna-Flora	Plant ⊑ ¬Animal

⊑ Fauna-Flora}}		
{**Ω' précédante** ∪ {{{Carnivorous-Plant ⊑ Animal} ∪ {Plant ⊑ ¬Animal}} ∪ {Carnivorous-Plant ⊑ Plant}}	Carnivorous-Plant ⊑ Plant	Carnivorous-Plant ⊑ Animal

A la fin de la deuxième itération nous avons Ω ← Ω'. Ainsi Ω comprend neuf sous-ontologies candidates :
Ω = {
Ω[1] : {{Carnivorous-Plant ⊑ Animal} ∪ {Animal ⊑ Fauna-Flora} ∪ {Plant ⊑ Fauna-Flora}}
Ω[2] : {{Carnivorous-Plant ⊑ Animal} ∪ {Animal ⊑ Fauna-Flora} ∪ {Carnivorous-Plant ⊑ Plant}}
Ω[3] : {{Carnivorous-Plant ⊑ Animal} ∪ {Animal ⊑ Fauna-Flora} ∪ {Plant ⊑ ¬Animal}}
Ω[4] : {{Carnivorous-Plant ⊑ Animal} ∪ {Carnivorous-Plant ⊑ Plant} ∪ {Animal ⊑ Fauna-Flora}}
Ω[5] : {{Carnivorous-Plant ⊑ Animal} ∪ {Carnivorous-Plant ⊑ Plant}} ∪ {Plant ⊑ Fauna-Flora}}
Ω[6] : {{Carnivorous-Plant ⊑ Animal} ∪ {Carnivorous-Plant ⊑ Plant}} ∪ {Plant ⊑ ¬Animal}}
Ω[7] : {{Carnivorous-Plant ⊑ Animal} ∪ {Plant ⊑ ¬Animal}} ∪ {Animal ⊑ Fauna-Flora}}
Ω[8] : {{Carnivorous-Plant ⊑ Animal} ∪ {Plant ⊑ ¬Animal}} ∪ {Plant ⊑ Fauna-Flora}}
Ω[9] : {{Carnivorous-Plant ⊑ Animal} ∪ {Plant ⊑ ¬Animal}} ∪ {Carnivorous-Plant ⊑ Plant}}
}

L'appel du raisonneur pour vérifier la cohérence de ces sous-ontologies candidates détecte une incohérence à Ω[9] =
{
{Carnivorous-Plant ⊑ Animal} ∪ {{Plant ⊑ ¬Animal} ∪ {Carnivorous-Plant ⊑ Plant}
}
La variable cohérence devient *false* et Ω[9] est retournée comme la sous-ontologie incohérente minimale.

II- EXECUTION DE L'ALGORITHME DE MISE EN CORRESPONDANCE ET SELECTION (CHAPITRE 5, TABLE 3) :

En prenant toujours le même exemple d'ontologie et de changement, et considérant le patron d'incohérences « disjonction liée à une subsomption » on aura en input :

Sous-ontologie incohérente
O : {{Carnivorous-Plant ⊑ Animal} ∪ {Plant ⊑ ¬Animal}} ∪ {Carnivorous-Plant ⊑ Plant}}
Patron d'incohérence
P_{inc} : IncP_Disjointness_Subsumption

L'application de cet algorithme fait intervenir deux parties de matching :
- dans la première partie :
  L'algorithme compare la correspondance des types des entités impliquées dans le patron d'incohérence considéré et celles de la sous-ontologie incohérente considérée (Matching_entité(e_p, e_o)) : nous avons une correspondance de 3 entités classe.

appariement_entité = {(SuperClass1, `Plant`), (SuperClass2, `Animal`), (SubClass, `Carnivorous-Plant`)}

- <u>dans la deuxière partie</u> :

-- L'algorithme compare le voisinage des entités appariées (Correspondance_voisinage((e_p, e_o))
Comme on a correspondance entre :
{`Carnivorous-Plant` ⊑ `Plant`} et {SubClass ⊑ SuperClass1}
{`Carnivorous-Plant` ⊑ `Animal`} et {SubClass ⊑ SuperClass2}

appariement_voisinage = {(SuperClass1, `Plant`), (SuperClass2, `Animal`), (SubClass, `Carnivorous-Plant`)}

-- Ensuite, l'algorithme compare les axiomes des entités appariées (Matching_axiome(*h*))
On vérifie la correspondance de tous les axiomes définis sur les classes du patron P_{inc} :
`IncP_Disjointness_Subsumption` avec les axiomes définis sur les classes de l'ontologie qui leur sont appariées
Ici on a l'axiome {SuperClass1 ⊑ ¬ SuperClass2} et {`Plant` ⊑ ¬`Animal`}

alignement = {{SubClass ⊑ SuperClass1} ∪ { SubClass ⊑ SuperClass2} ∪ {SuperClass1 ⊑ ¬ SuperClass2}}

on retrouve à la fin l'ensemble des axiomes définissant l'interface du patron P_{inc} : `IncP_Disjointness_Subsumption` ce qui permet de le sélectionner.

Notons que les trois méthodes Matching_entité(e_p, e_o), Correspondance_voisinage((e_p, e_o) et Matching_axiome(*h*) sont dans leur détails spécifiques à chaque interface du patron considéré dans la mise en correspondance (ses arguments (leur type, signification, etc.) et ses axiomes)

Résumé

Les travaux de recherche développés dans cette thèse, définissent une approche d'évolution d'ontologie *Onto-Evo^{a}l* (Ontology Evolution-Evaluation) qui s'appuie sur une modélisation de patrons de gestion de changement CMP (Change Management Patterns). Ces patrons spécifient des classes de changements, des classes d'incohérences et des classes d'alternatives de résolution. Sur la base de ces patrons et des relations sémantiques entre eux, un processus automatisé permettant de conduire l'application des changements tout en maintenant la cohérence de l'ontologie évoluée a été développé. L'approche intègre également une activité d'évaluation basée sur un modèle de qualité d'ontologie qui a été défini. Ce modèle est employé pour guider la gestion des incohérences en évaluant l'impact des résolutions proposées sur le contenu et l'usage de l'ontologie à travers un ensemble de métriques quantitatives et ce, afin de choisir une résolution qui préserve la qualité de l'ontologie évoluée.

La gestion des changements étant fortement liée au modèle dans lequel est représentée l'ontologie, nous nous sommes focalisés sur le langage OWL en tenant compte de l'impact des changements sur la cohérence logique de l'ontologie telle que spécifiée dans la couche OWL DL.

Les principales contributions de l'approche résident dans la modélisation des patrons de gestion de changement guidant le processus d'évolution, l'intégration de l'évaluation de la qualité pour optimiser la résolution des changements et la modélisation formelle et explicite du journal d'évolution.

Mots-clés :

Evolution d'ontologie, Gestion de changements OWL DL, Modélisation de patrons, Résolution d'incohérences logiques, Evaluation de qualité d'ontologie.

Abstract

The research developed in this thesis defines an ontology evolution approach *Onto-Evo^{a}l* (Ontology Evolution-Evaluation) guided by Change Management Pattern (CMP) modeling. CMP patterns specify three categories of classes: Change, Inconsistency and Resolution Alternative. Based on the modeled patterns and the semantic relations between them, we propose an automated process driving change application while maintaining consistency of the evolved ontology. In addition, the approach integrates an evaluation activity supported by an ontology quality model that we have defined. The quality model is used to guide inconsistency resolution by assessing the impact of the resolutions proposed by the evolution process on ontology quality and selecting the resolution that preserves the quality of the evolved ontology.

Change management depends closely on the ontology representation model, we focus on OWL language and we consider change impact on logical consistency as specified in OWL DL layer.

The main contributions of the approach are: Change Management Patterns guiding the ontology evolution process; the use of quality evaluation to optimize change resolution; and a formal and explicit modeling of the evolution log.

Keywords:

Ontology Evolution, OWL DL Change Management, Pattern Modeling, Logical Inconsistency Resolution, Ontology Quality Evaluation.