Gestion de l'évolution d'un Web sémantique d'entreprise

Phuc Hiep Luong

ED 84 : Sciences et Technologies de l'Information et de la Communication

N°attribué par la bibliothèque
|__|__|__|__|__|__|__|__|__|__|

T H È S E

pour obtenir le grade de
Docteur de l'Ecole des Mines de Paris
Spécialité "Informatique Temps Réel, Robotique, Automatique"

présentée et soutenue publiquement par
Phuc Hiep LUONG

le 14 décembre 2007

GESTION DE L'ÉVOLUTION D'UN WEB SÉMANTIQUE D'ENTREPRISE

Directrice de thèse : Rose Dieng-Kuntz

Jury

M.	Nhan Le Thanh	Président
M.	Gilles Kassel	Rapporteur
Mme	Chantal Reynaud	Rapporteur
Mme	Sylvie Després	Examinatrice
Mme	Rose Dieng-Kuntz	Examinatrice
M.	Alain Boucher	Examinateur (Co-encadrant)

À mes parents
À ma femme et à Tôm

Remerciements

Il m'est particulièrement agréable de pouvoir remercier ici Mme. Rose Dieng-Kuntz, Directrice de l'équipe ACACIA/EDELWEISS et directrice de cette thèse, pour avoir bien voulu m'accueillir dans son équipe de recherche et m'avoir encadré durant ces trois années. Je lui exprime ma profonde gratitude. Sa gentillesse et ses encouragements me furent d'un constant appui sans lequel la persévérance m'aurait manqué. Je tiens aussi à remercier M. Alain Boucher, Professeur à l'Institut de la Francophonie pour l'Informatique et mon co-directeur de thèse, pour sa disponibilité et son soutien et d'avoir permis que ma thèse se déroule dans les meilleures conditions.

Mes respects et ma gratitude vont également aux membres du jury qui m'ont fait l'honneur de juger ce travail. Je remercie tout d'abord M. Nhan Le Thanh, Professeur à l'université de Nice Sophia Antipolis, pour avoir présidé le jury de cette thèse.

Je remercie M. Gilles Kassel, Professeur à l'université de Picardie Jules Verne ainsi que Mme. Chantal Reynaud, Professeur à l'université Paris XI pour avoir accepté de rapporter ce manuscrit, ainsi que pour l'intérêt qu'ils ont manifesté à l'égard de ce travail de thèse. Je remercie également Mme. Sylvie Després, Maître de conférence à l'université Paris-Nord, d'avoir accepté d'examiner mes travaux.

J'adresse mes sincères remerciements aux mes collègues de l'équipe ACACIA/EDELWEISS : Olivier Corby, Alain Giboin, Fabien Gandon, Khaled Khelif, Leila Khelif, Angéla Ouvrier, Hacène Cherfi, Amira Tifous, Mohamed Bennis, Michel Buffa, Noureddine Mokhtari, Emmanuel Jamin, Adil El_Ghali, Freddy Limpens; et aux anciens membres : Virginie Bottollier, Patricia Maleyran, Thanh Le Bach, Tuan Dung Cao, Sylvain Dehors, Laurent Alamarguy.

Je tiens à remercier l'Institut de la Francophonie pour l'Informatique (IFI, Vietnam) et les membres du laboratoire MSI de m'avoir accueilli pendant le déroulement de la thèse au Vietnam. Je remercie l'Agence Universitaire de la Francophonie (AUF) et l'INRIA Sophia Antipolis pour avoir financé en partie ces travaux.

La rédaction de la thèse nécessite le courage et le soutien de Priscille Durville. Je veux lui exprimer sincèrement ma gratitude pour son travail de relecture ainsi que son aide qui a fait avancer mon niveau de français. J'aimerais aussi remercier Priscille et Duc Bao Le pour leur collaboration dans certains travaux d'implémentation du prototype de ma thèse.

Je souhaite exprimer mon amitié au “bande” de doctorants à l’INRIA : Van Thinh Vu, Lan Le Thi, Thanh Trung Nguyen, Viet Dung Doan, Anh Tuan Nghiem pour les discussions infinies au déjeuner et également aux étudiants de l’AJVN (Association des Jeunes Vietnamiens à Nice) qui m’ont partagé les événements inoubliables pendant mes années dans la région de Côte d’Azur.

Mes remerciements les plus chaleureux vont aux membres de ma famille: mes parents, mes beaux-parents, mon frère et ma sœur pour leur encouragement infini. Et spécialement, mon petit fils Tôm, qui m’a beaucoup donné de l’énergie pendant ces années de travail, malgré la distance géographique qui nous a parfois séparés. Enfin, tout au fond de mon cœur, merci à ma femme Ngọc, pour sa patience, son amour, sa confiance et son soutien constant à travers ces longues années. Sans elle, rien de tout cela n’aurait été possible.

Résumé

Le Web Sémantique d'Entreprise (WSE) est une approche particulière de la Gestion des Connaissances d'une entreprise pour la prochaine génération du Web Sémantique dans laquelle on donne à une information un sens bien défini pour permettre aux ordinateurs et aux utilisateurs de travailler en coopération. Dans la réalité, les organisations vivent dans un environnement hétérogène, dynamique et en cours d'évolution qui mène souvent à des changements externes et internes requérant l'évolution de leur système de gestion des connaissances. Peu de recherches actuelles font face aux changements et fournissent des facilités de maintenance pour le système de gestion des connaissances. L'objectif de cette thèse est de contribuer à lever cette limitation.

Dans ce manuscrit, nous présentons une nouvelle approche de la gestion de l'évolution du WSE. Nous nous focalisons en particulier sur l'évolution de l'ontologie et de l'annotation sémantique qui sont deux composants importants du WSE. Nous nous intéressons à deux scénarios d'évolution de l'ontologie : (i) avec trace et (ii) sans trace de changements ontologiques effectués. Ces deux scénarios sont fréquents dans les situations réelles et ils peuvent entraîner des inconsistances au niveau des annotations sémantiques reposant sur cette ontologie modifiée. Pour chacun des contextes d'évolution, nous développons des approches équivalentes : une approche procédurale et une approche basée sur des règles en vue de gérer l'évolution des annotations sémantiques et, en particulier, de détecter et de corriger les annotations sémantiques inconsistantes.

Ces propositions ont été implémentées et validées dans le système CoSWEM qui facilite la gestion de l'évolution du WSE en permettant de réaliser certaines tâches d'une manière automatique ou semi-automatique telles que la comparaison d'ontologies différentes, la détection et la correction des inconsistances sur les annotations sémantiques, etc. Ce système a été expérimenté dans le cadre des projets PALETTE et E-WOK_HUB sur un ensemble de données réelles et évolutives provenant de ces projets.

Mots-clés : web sémantique, évolution, web sémantique d'entreprise, ontologie, annotation sémantique, règles, détection d'inconsistance, résolution d'inconsistance, stratégie de résolution, RDF(S)

Abstract

Corporate Semantic Web (CSW) is an approach of the Knowledge Management in an organisation for the next generation of the Semantic Web in which information is given well-defined meaning, better enabling computers and people to work in co-operation. Actually, the organizations live in a heterogeneous, dynamic and evolving environment which often leads to internal and external changes requiring the evolution of their knowledge management system. Few current researches face the change management problem and provide the maintenance facilities for the knowledge management system. The objective of this thesis is to contribute to reducing this limitation.

In this dissertation, we present a new approach for the evolution management of the CSW. We focus particularly on the ontology evolution and semantic annotation evolution which are two important components of the CSW. We are interested in two main scenarios of ontology evolution : (i) with trace and (ii) without trace of ontology changes which are carried out during its evolution. These two scenarios are often encountered in reality and they lead to inconsistencies of the annotations semantics using this modified ontology. Corresponding to each context of evolution, we have developed equivalent approaches: a procedural approach and a rule-based approach in order to manage semantic annotations evolution and particularly to detect inconsistent annotations and to guide the process of solving these inconsistencies.

These propositions were implemented and validated in the CoSWEM system which facilitates the evolution management of the CSW. It enables to carry out some tasks automatically or semi-automatically such as comparison of different ontologies, inconsistency detection and correction of the semantic annotations, etc. This system was also experimented within the framework of the industrial projects PALETTE and E-WOK_HUB with a set of real and evolving data from these projects.

Keywords : semantic web, evolution, corporate semantic web, ontology, semantic annotation, rules, inconsistency detection, inconsistency resolution, resolution strategy, RDF(S).

Table des matières

Résumé ..... **iii**

Abstract ..... **v**

Table des matières ..... **vii**

Liste des figures ..... **xi**

Liste des tableaux ..... **xv**

Liste des tableaux ..... **xv**

Introduction **1**

1 État de l'art **13**

1.1 Web sémantique 14

1.1.1 Architecture et langages du Web sémantique 15

1.1.2 Composants principaux du Web sémantique 24

1.1.3 De la Mémoire d'entreprise au Web sémantique d'entreprise 35

1.2 Évolution du système de gestion des connaissances 36

1.2.1 Problème de changements et de l'évolution 36

1.2.2 Approches principales sur la gestion de l'évolution 38

1.3 Conclusion 45

2 Évolution du Web Sémantique d'Entreprise **47**

2.1 Introduction 48

2.1.1 Web sémantique d'entreprise - Approche Acacia 48

2.1.2 Cycle de développement 50

2.1.3 Applications du Web Sémantique d'entreprise 51

2.2 Problèmes d'évolution du WSE 52

2.2.1 Causes possibles des problèmes d'évolution 54

2.2.2 Scénarios d'évolution 57

2.3 Aspects importants de l'évolution de l'ontologie 61

2.3.1 Processus d'évolution 61
2.3.2 Représentation des changements 62
2.3.3 Détection des changements ontologiques 64
2.3.4 Vérification et résolution des inconsistances 65
2.3.5 Comparaison et Gestion des versions de l'ontologie 67
2.3.6 Discussion 68
2.4 Aspects importants de l'évolution de l'annotation sémantique 69
2.4.1 Analyse des effets des changements ontologiques sur l'annotation 70
2.4.2 Détection des annotations inconsistantes 71
2.4.3 Résolution des annotations inconsistantes 71
2.4.4 Discussion 73
2.5 Conclusion 73
3 CoSWEM – un système de gestion de l'évolution du WSE 77
3.1 Motivations 78
3.2 Architecture du système CoSWEM 79
3.2.1 Architecture 79
3.2.2 Description des composants du système 80
3.2.3 Intégration de CoSWEM dans un Web sémantique d'entreprise ...83
3.3 Évolution de l'ontologie dans le système CoSWEM 83
3.3.1 Propagation des changements de l'ontologie 84
3.3.2 Principaux aspects de l'évolution de l'ontologie 85
3.4 Évolution de l'annotation sémantique dans le système CoSWEM 86
3.4.1 Étapes principales 87
3.4.2 Approches de détection et de résolution des inconsistances 88
3.5 Outils de support utilisés dans le système CoSWEM 91
3.5.1 CORESE - Moteur de recherche sémantique 91
3.5.2 ECCO – Editeur Collaboratif et Contextuel d'Ontologie 93
3.5.3 SemTag - Librairie de tags JSP sémantiques 94
3.6 Conclusion 96
4 Évolution de l'ontologie 97

4.1 Démarche générale 98
4.1.1 Modèle de l'ontologie 99
4.1.2 Processus d'évolution de l'ontologie 99
4.2 Représentation des changements de l'ontologie 100
4.2.1 Classification des changements ontologiques 100
4.2.2 Construction de l'ontologie d'évolution 105
4.2.3 Trace de changement 110
4.3 Résolution des changements de l'ontologie 113
4.3.1 Niveau d'inconsistance 113
4.3.2 Stratégies de résolution 114
4.3.3 Résolution des inconsistances de l'ontologie 118
4.3.4 Gestion et comparaison des versions de l'ontologie 120
4.4 Propagation des changements de l'ontologie 123
4.4.1 Propagation aux ontologies dépendantes 124
4.4.2 Propagation aux annotations sémantiques 124
4.5 Conclusion 125
5 Évolution de l'annotation sémantique dans le contexte de modification de l'ontologie 127
5.1 Description du problème 128
5.1.1 Exemple de motivation 128
5.1.2 Hypothèses 130
5.1.3 Définitions 130
5.1.4 Les approches de l'évolution de l'annotation sémantique 133
5.2 Approche procédurale - évolution de l'annotation sémantique avec trace de changements 138
5.2.1 Processus d'évolution avec la trace de changement 138
5.2.2 Approche procédurale 139
5.3 Approche basée sur des règles - évolution de l'annotation sémantique sans trace de changements 144
5.3.1 Processus d'évolution sans trace de changement 144
5.3.2 Détection des inconsistances de l'annotation sémantique 145
5.3.3 Correction des inconsistances de l'annotation sémantique 149

5.4 Jeu de tests et exemples.....153
5.5 Conclusion.....158
6 Implémentation et Évaluation du CoSWEM.....161
6.1 Implémentation du système.....162
6.1.1 Architecture d'implémentation.....162
6.1.2 Conception du système.....164
6.2 Expérimentations.....169
6.2.1 Description des projets de l'expérimentation.....170
6.2.2 Expérimentations.....173
6.3 Évaluation des résultats.....175
6.3.1 Exigences des fonctionnalités.....175
6.3.2 Mesures d'évaluation.....176
6.3.3 Évaluation des résultats.....178
6.3.4 Discussion.....187
6.4 Conclusion.....190
Conclusion.....191
Bibliographie.....199
Annexe A - Stratégies de résolution.....209
Annexe B – Ontologie d'évolution CoSWEM.....235
Annexe C – Exemple d'une trace d'évolution.....249

Liste des figures

Figure 1 - Le web du futur [Spivack, 2004] 14
Figure 2 - Architecture du Web sémantique [Berners-Lee et al., 2001] 16
Figure 3 - Modèle de triplet en RDF 20
Figure 4 - Exemple d'un extrait de l'ontologie..... 26
Figure 5 – Rôle des annotation dans le système de gestion des connaissances 30
Figure 6 - Utilisation de l'annotation sémantique..... 31
Figure 7 - Evolution du Web actuel vers le Web sémantique 46
Figure 8 - Architecture d'un Web sémantique d'entreprise [Dieng et al., 2004] 49
Figure 9 - Le cycle de vie de la mémoire d'entreprise [Dieng et al., 2001]..... 50
Figure 10 - Contexte de changements 53
Figure 11 - Changements des composants du WSE 54
Figure 12 - Un extrait de l'ontologie O'CoP avant les changements 59
Figure 13 - Exemple d'une annotation basée sur l'ontologie O'CoP..... 59
Figure 14 - Un extrait de l'ontologie O'CoP après les changements 60
Figure 15 - Architecture du système CoSWEM..... 80
Figure 16 - Intégration de CoSWEM dans un Web sémantique d'entreprise 83
Figure 17 – Propagation des changements de l'ontologie dans le système CoSWEM..... 84
Figure 18 - Approche procédurale pour l'évolution avec le trace de changement 89
Figure 19 - Approche de gestion des versions d'ontologie 90
Figure 20 - Approche basée sur des règles pour l'évolution sans trace de changement 90
Figure 21 - Architecture de Corese 92
Figure 22 - Interface de l'éditeur d'ontologie ECCO 94
Figure 23 - Architecture de SeWeSe 95
Figure 24 - Exemple d'un tag de la librairie SemTag 96
Figure 25 - Processus d'évolution de l'ontologie 99
Figure 26 - Les voisins du concept `Member_role`..... 102
Figure 27 - Exemple d'un changement entraînant la correction obligatoire..... 104
Figure 28 - Exemple d'un changement entraînant la correction facultative 104
Figure 29 - Extrait de l'ontologie d'évolution CoSWEM 108

Figure 30 - Représentation d'un extrait de l'ontologie d'évolution CoSWEM 108
Figure 31 - Un changement ontologique et ses paramètres.....109
Figure 32 - Les concepts et les propriétées de l'ontologie d'évolution.....110
Figure 33 - Extrait d'une trace de changements 111
Figure 34 - Exemple d'une trace de changement de l'ontologie 112
Figure 35 - Stratégies de résolution pour l'ontologie 114
Figure 36 - Interface de l'édition de l'ontologie dans l'éditeur ECCO.....118
Figure 37 - Hiérarchie de concepts dans l'éditeur ECCO.....119
Figure 38 - Comparaison des lignes de texte supprimés entre deux versions.....120
Figure 39 - Comparaison des versions de l'ontologie de l'outil OntoView.....121
Figure 40 - Comparaison des concepts différents entre deux versions de l'ontologie par l'outil CoSWEM.....121
Figure 41 - Comparaision des versions de l'ontologie dans CoSWEM 123
Figure 42 - Réutilisation des ontologies et les ontologies dépendantes.....124
Figure 43 - Propagation du changement de l'ontologie vers l'annotation sémantique 125
Figure 44 - Extrait de l'ontologie et les triplets d'annotations avant la modification.....128
Figure 45 - Extrait de l'ontologie et les triplets d'annotations après la modification.....129
Figure 46 - Processus d'évolution pour l'approche procédurale.....138
Figure 47 - Changements de l'ontologie effectués 139
Figure 48 – Exemple d'une trace d'évolution.....140
Figure 49 - Exemple d'une requête envoyée à Corese en langage SPARQL.....141
Figure 50 - Résultat d'une requête de recherche effectuée par Corese 142
Figure 51 - Annotations inconsistantes trouvées concernant le concept supprimé `personnes_physique3`.....143
Figure 52 - Processus d'évolution pour l'approche basée sur des règles 144
Figure 53 - Algorithme de détection des inconsistances sur les concepts 147
Figure 54 - Exemple des triplets inconsistants détectés.....147
Figure 55 - Représentation d'une règle de détection d'inconsistance.....148
Figure 56 - Exemple des triplets corrigés consistants.....150
Figure 57 - Algorithme de recherche d'un « meilleur » concept 151
Figure 58 - Représentation d'une règle de correction d'inconsistance.....152
Figure 59 - Architecture d'implémentation du système CoSWEM.....162
Figure 60 - Modèle global et diagramme de séquence du MVC.....165
Figure 61 - Modèle MVC appliqué dans CoSWEM.....166
Figure 62 - Conception des règles.....167
Figure 63 - Extrait du fichier de projection des règles (rule-mapping.xml) 168

Figure 64 - Application des règles et présentation des résultats 169
Figure 65 - Décomposition des lots du projet PALETTE 171
Figure 66 - Ensemble de portails web du projet E-WOK_HUB 172
Figure 67 – Mesures utilisées pour évaluer les résultats détectés 178
Figure 68 - Certains concepts et propriétés principaux de l'ontologie COG 179
Figure 69 - Liste des annotations inconsistantes contenant le concept renommé `Canton` 181
Figure 70 - Certains concepts et propriétés principaux de l'ontologie O'CoP 182
Figure 71 - Liste des concepts inconsistants détectés 183
Figure 72 - Liste des propriétés inconsistantes détectées 184
Figure 73 - Liste des inconsistances de domaine 184
Figure 74 – Diagramme de nombre des triplets inconsistants détectés 185
Figure 75 – Diagramme du résultat des mesures P, R et F-mesure 186
Figure 76 - Interface de correction des triplets inconsistants 187

Liste des tableaux

Tableau 1 - Comparaison de certaines approches sur l'évolution de l'ontologie 68
Tableau 2 – Comparaison de certaines approches sur l'évolution de l'annotation 73
Tableau 3 - Liste des changements élémentaires 101
Tableau 4 - Liste des changements composites 103
Tableau 5- Description d'un changement 103
Tableau 6 - Classification des changements entraînant la correction obligatoire et facultative 104
Tableau 7 - Stratégies de résolution pour le changement DeleteConcept 115
Tableau 8 - Stratégies de résolution pour l'ontologie et l'annotation sémantique 135
Tableau 9 - Jeu de tests pour les annotations sémantiques 153
Tableau 10 - Description des tags XML du ficher de règles (rule.xml) 167
Tableau 11 - Description des tags XML du ficher de projection (rule-mapping.xml) 168
Tableau 12- Résultat de la détection des annotations inconsistantes 180
Tableau 13 - Nombre de triplets inconsistants détectés sur chaque élément 185
Tableau 14 – Résultat des mesures utilisées pour la validation 186

Introduction

Le terme ***Évolution*** est défini dans le Dictionnaire de l'Académie Française[1] (9ème édition, dictionnaire en ligne) comme une *"suite de transformations progressives, de développements graduels, de modifications"*. Il s'agit de changements et de modifications ayant lieu d'une manière progressive. Le sujet de l'évolution joue un rôle important depuis longtemps dans les recherches de plusieurs domaines tels que l'économie, la génétique, la psychologie, les sciences du langage, etc.

Dans le domaine de l'informatique, l'évolution est aussi un sujet qui attire beaucoup de travaux de recherche. Il existe des études sur l'évolution du système de base de données, l'évolution en génie logiciel, l'évolution des bases de connaissances, etc. Dans la nouvelle génération du Web, le Web sémantique favorisera l'évolution de la *connaissance humaine* vers un autre type de *connaissance "compréhensible"* par la machine qui peut être traitée automatiquement dans les systèmes de gestion des connaissances. L'application des technologies du Web sémantique (i.e. l'ontologie, l'annotation sémantique, la recherche intelligente…) aux systèmes de gestion des connaissances peut entraîner l'évolution de ces systèmes et elle doit être bien gérée. Ainsi, ce mémoire de thèse s'intéresse aux études sur l'évolution du Web sémantique et particulièrement sur les deux composants importants, i.e. l'ontologie et l'annotation sémantique.

Contexte scientifique

À l'heure actuelle, les sources d'informations et de connaissances dans les entreprises (ou organisations) deviennent de plus en plus importantes et elles jouent aussi un rôle crucial pour le développement de ces entreprises. Le partage du travail nécessite un partage de connaissances au sein des entreprises et ce besoin peut devenir encore plus crucial. Pour réussir, les entreprises doivent bien gérer ces sources de connaissances et supporter l'utilisation effective des connaissances. L'intégration d'un système de gestion des connaissances permet à une entreprise d'offrir un accès global à l'ensemble des sources d'informations et aide à améliorer le partage des connaissances dans l'entreprise.

[1] http://atilf.atilf.fr/academie9.htm

Dans les dernières années, le développement impressionnant de l'Internet a renforcé l'apparition d'une énorme quantité d'informations disponibles sur le Web. La richesse et la croissance exponentielle de ce volume d'informations promet d'être une mine d'or pour les entreprises. Cependant, cette croissance d'informations donnera lieu aussi à de vrais obstacles si les informations ne sont pas bien structurées et bien représentées. Cela montre les limitations du Web actuel du point de vue de la recherche, l'organisation, l'accès et la maintenance de ces informations… selon les exigences de l'utilisateur.

La naissance de la nouvelle génération du Web (i.e. le Web sémantique) est un des efforts pour pallier à ces limitations. L'idée du ***Web sémantique*** est de modéliser le contenu des ressources du Web en ajoutant de la sémantique sous forme de méta-données en vue de rendre les ressources compréhensibles par des machines [Berners-Lee et al., 2001]. Autrement dit, il s'agit de décrire ces ressources selon une représentation formelle avec une sémantique clairement définie et qui soit conçue pour une interprétation par des programmes. La base de l'infrastructure du Web sémantique s'appuie sur l'explicitation de la conceptualisation d'un domaine, partagée par une communauté et représentée dans une ontologie du domaine concerné.

L'intégration des technologies du Web sémantique dans les systèmes de gestion des connaissances est une des approches qui apportent de nouvelles perspectives. L'approche de l'équipe ACACIA/EDELWEISS [2] repose sur la combinaison des technologies du Web sémantique et les moyens de communication de l'entreprise (i.e. l'intranet, l'intraweb) pour caractériser et construire une mémoire d'entreprise. A partir de l'analogie entre les ressources du Web et les ressources de la mémoire d'entreprise, [Dieng-Kuntz, 2005] a proposé de matérialiser une mémoire d'entreprise sous forme d'un ***Web Sémantique d'entreprise*** (ou Web sémantique d'organisation) constitué (1) de ressources (documents, personnes, services), (2) d'ontologies (décrivant le vocabulaire conceptuel partagé par une ou plusieurs communautés dans l'entreprise) et (3) d'annotations sémantiques sur les ressources (e.g. sur les compétences des personnes, le contenu sémantique des documents ou les caractéristiques des services…).

En réalité, les organisations vivent dans un environnement hétérogène, dynamique et en cours d'évolution qui mène souvent à des changements externes et internes constituant l'évolution de leur système de gestion des connaissances du Web sémantique d'entreprise. Une des approches de recherche actuelles sera d'approfondir le cycle de vie d'un tel Web Sémantique d'entreprise et les problèmes liés à son évolution : l'évolution de chacun des composants (ontologies, ressources, annotations) et l'évolution des liens entre ces composants. Parmi ces composants, ***l'ontologie*** et ***l'annotation sémantique*** jouent un rôle important dans un Web sémantique d'entreprise. Les ontologies sont nécessaires pour les

[2] http://www-sop.inria.fr/edelweiss/ (successeur de http://www-sop.inria.fr/acacia/)

représentations sémantiques nécessitant un certain consensus de communauté. Les annotations sémantiques consistent à ajouter des méta-données structurées aux ressources documentaires du Web mais aussi des intranets des entreprises. Cependant, ces deux composants sont souvent modifiés et évoluent afin de s'adapter aux nouveaux besoins ou exigences. Par exemple, les ontologies doivent normalement changer pour répondre aux types de changements sur le domaine, sur la conceptualisation ou sur la spécification explicite. Plusieurs problèmes apparaissent quand on essaie d'utiliser ensemble des ontologies développées indépendamment, ou quand des ontologies existantes sont adaptées pour répondre à de nouveaux objectifs. D'autre part, les modifications de l'ontologie de référence pourraient aussi entraîner des changements continus dans les annotations sémantiques.

La majorité des recherches existantes dans le domaine des systèmes de gestion des connaissances, de l'ontologie ou des annotations sémantiques sont centrées principalement sur des problèmes de construction. Peu de recherches font face aux changements et fournissent des facilités de maintenance pour le système de gestion des connaissances. L'objectif de notre recherche est de contribuer à lever cette limitation. C'est pourquoi, dans notre travail de thèse, nous nous intéressons à la ***gestion de l'évolution du Web sémantique d'entreprise*** et particulièrement des deux composants que sont l'ontologie et l'annotation sémantique.

Les travaux présentés dans ce manuscrit sont réalisés dans le cadre de recherches de l'équipe ACACIA/EDELWEISS dont les thèmes principaux reposent sur l'application des modèles, des méthodes et des techniques de la gestion des connaissances (e.g. création, gestion, distribution, évaluation, évolution, etc.) pour une organisation ou une communauté. Ils s'appuient également sur des problématiques concrètes à résoudre dans le cas de différents projets de recherches listés ci-dessous.

Projets de recherches

- **Projet PALETTE** [3] (Pedagogically sustained Adaptive LEarning Through the exploitation of Tacit and Explicit knowledge) **:** Ce projet européen vise à faciliter et augmenter l'apprentissage individuel et organisationnel dans les Communautés de Pratique (CoPs). Nous nous intéressons au lot *WP3-Services de Gestion des connaissances* dont l'objectif est de faciliter la génération, l'utilisation et la gestion des connaissances liées aux activités de CoPs. Les tâches principales de ce lot sont d'élaborer des ontologies décrivant les connaissances et activités de CoPs (i.e. ontologie O'CoP) et de développer des services orientés vers la gestion des connaissances de CoPs. L'environnement distribué et collaboratif de CoPs (e.g. une ontologie est créée et

[3] http://palette.ercim.org/

modifiée par différents partenaires) nécessite aussi les études sur la gestion de l'évolution de la mémoire de CoPs après les modifications.

- **Projet E-WOK_HUB**[4] (Environmental Web Ontology Knowledge Hub) **:** Le projet ANR E-WOK_HUB a pour objectif d'exploiter les technologies du Web sémantique pour développer des systèmes opérationnels autorisant la coopération sur internet entre différentes organisations (entreprises, instituts, ...) impliquées dans un workflow d'ingénierie. Et ce, en mettant en place un ensemble de portails communicants proposant à la fois des applications web accessibles à des utilisateurs finaux et des services web accessibles aux applications métiers. Ces portails sont la brique de base d'une architecture dédiée à l'exploitation des techniques de traitement des données et des connaissances. Concernant notre travail de thèse, nous nous sommes intéressés au scénario applicatif et aux outils de support (i.e. l'éditeur d'ontologie ECCO) du projet E-WOK_HUB.

Problématique et Questions de recherche

Jusqu'à maintenant, les études sur l'évolution des connaissances restent encore un sujet de recherche qui attire beaucoup l'attention des chercheurs dans le domaine de l'Intelligence Artificielle. En particulier, dans la prochaine génération du Web sémantique, les caractéristiques principales telles que l'environnement dynamique, collaboratif, hétérogène et distribué… renforceront clairement le problème des changements et de l'évolution. Ce dernier doit être bien géré et contrôlé dans la nouvelle infrastructure du Web sémantique.

Dans ce manuscrit, nous nous intéressons aux processus d'évolution de l'ontologie et de l'annotation sémantique. Concrètement, nous étudions plus particulièrement les cas où la modification de l'ontologie peut affecter les annotations sémantiques. Nous proposons des réponses pour les questions de recherche suivantes :

Question 1 : Quels sont les types de changements possibles et leurs affectations au système de gestion des connaissances ?

Il s'agit d'avoir une vue d'ensemble des différents types de changements possibles qui peuvent avoir lieu au sein d'un système de gestion des connaissances durant son cycle de vie. Dans le cadre du Web sémantique d'entreprise, les changements à étudier sont ceux qui interviennent sur ses deux principaux composants à priori : les ontologies et les annotations sémantiques. Etant donné que les ontologies doivent s'adapter en permanence aux nouveaux besoins de l'environnement, elles sont sujettes à modifications. Nous nous focalisons donc sur les différents types de changements de l'ontologie ainsi que sur l'impact que ces changements peuvent avoir tant au niveau ontologique qu'au niveau des

[4] http://www-sop.inria.fr/acacia/project/ewok/index.html

annotations. Plus concrètement, nous nous intéressons à la classification des changements ontologiques selon certains critères (i.e. le niveau d'abstraction, l'affectation sur l'annotation…), la représentation, la résolution et la propagation de ces changements.

Il est à noter que l'exécution des changements de l'ontologie peut entraîner des inconsistances dans les autres parties de la même ontologie, dans les ontologies dépendantes ou dans les annotations sémantiques concernées. Si ces inconsistances ne sont pas détectées et résolues, elles empêcheront l'utilisation et le fonctionnement du système de gestion des connaissances.

Question 2 : Comment peut-on détecter les inconsistances générées à cause des changements de l'ontologie?

Quand une ontologie est développée par plusieurs ontologistes d'une manière collaborative et qu'elle devient de plus en plus grande et complexe au niveau de la structure et de la représentation, il est difficile pour un ontologiste de comprendre toutes les parties de l'ontologie. Il est aussi difficile pour l'ontologiste de connaître les parties affectées (i.e. les éléments relatifs, les parties dépendantes, les annotations concernées, etc.) lorsque sont effectués des changements sur cette ontologie. Par conséquent, nous avons besoin d'un mécanisme qui surveille comment les changements ontologiques ont été effectués et détecte *automatiquement* les inconsistances possibles générées à cause de la modification de l'ontologie. Ce mécanisme nous permettra également de trouver les éléments affectés eux-mêmes (i.e. les annotations sémantiques inconsistantes) au cours de l'évolution.

Question 3 : Comment résoudre les inconsistances détectées à cause de la modification de l'ontologie pour assurer la consistance globale du système?

Nous avons mentionné le besoin de détecter les inconsistances produites par la modification de l'ontologie. Afin de rétablir la consistance du système de gestion des connaissances, il ne suffit pas de vérifier si l'ontologie reste encore cohérente avec le reste du système après ces changements, mais il faut résoudre les inconsistances sur l'ontologie elle-même et sur les parties dépendantes concernées (i.e. les annotations sémantiques utilisant cette ontologie). Nous considérons les deux scénarios principaux de l'évolution de l'ontologie : (i) avec trace de changements et (ii) sans trace de changements de l'ontologie. Correspondant à chaque scénario, nous devons proposer des approches convenables en vue de corriger les inconsistances et de rétablir l'utilisation cohérente de chaque composant au sein du système.

Question 4 : Comment appliquer et exploiter ces recherches sur l'évolution à un problème réel qui demande une gestion de l'évolution? Est-il possible de développer des outils d'aide pour gérer l'évolution du système de gestion des connaissances?

La gestion des changements et de l'évolution joue un rôle très important dans l'organisation. Nous voudrions informatiser ce processus dans le but d'améliorer l'efficacité de ce travail. Un système informatique ou des outils de support permettront à l'organisation de gérer l'évolution des changements d'une manière plus exacte, rapide et efficace. Les recherches proposées ci-dessus ont besoin d'être implémentées en des outils d'aide pour faciliter la détection et la résolution *automatique (ou semi-automatique)* des inconsistances ou permettre d'assister l'utilisateur en lui proposant des suggestions pour réaliser des tâches spécifiques.

Contributions et Approches

Le travail présenté dans ce manuscrit ne prétend pas fournir une solution complète répondant à tous les aspects différents du problème d'évolution du Web sémantique. Toutefois, ce travail a apporté un certain nombre de contributions au champ de recherches sur la gestion de l'évolution :

- Nous présentons une classification des changements de l'ontologie tels que les changements élémentaires ou composites, les changements entraînant les corrections obligatoires ou facultatives, les changements sur le concept ou sur la propriété… selon différents critères (i.e. les niveaux d'abstraction, l'affectation sur les annotations sémantiques, sur l'élément de l'ontologie). Cette classification aidera la représentation des changements ontologiques d'une manière plus formelle et déclarative.

- Nous proposons de formaliser la trace des changements effectués sur l'ontologie en construisant une ontologie d'évolution qui décrit formellement les types de changements ontologiques possibles ainsi que les informations concernant l'exécution du changement. Cette trace est générée par un éditeur d'ontologie.

- Les changements effectués sur l'ontologie peuvent entraîner des inconsistances sur les autres composants et les parties concernées (i.e. les annotations sémantiques). Nous avons approfondi deux scénarios d'évolution de l'ontologie (i) avec trace et (ii) sans trace de changement de l'ontologie. Nous avons proposé des approches correspondantes pour détecter les annotations sémantiques qui deviennent obsolètes et inconsistantes à cause de l'évolution de l'ontologie.

- Les inconsistances générées de l'ontologie et de l'annotation sémantique peuvent être résolues automatiquement grâce aux stratégies de résolution définies dans le cadre de notre travail. Nous avons construit un ensemble de stratégies de résolution pour l'ontologie et pour l'annotation sémantique respectivement. Ces

stratégies contiennent des solutions qui guident le processus de résolution pour tous les types de changements classifiés ci-dessus.

- Pour l'évolution de l'annotation sémantique, nous avons appliqué deux approches : (i) une approche procédurale et (ii) une approche basée sur des règles qui correspondent aux deux scénarios d'évolution dans lesquels on peut garder ou ne pas garder la trace des changements effectués.

- Finalement, nous avons appliqué les résultats de ces travaux dans le cadre de projets de recherche. Nous avons développé un système de gestion de l'évolution CoSWEM (Corporate Semantic Web Evolution Management) [Luong et Dieng-Kuntz, 2007] et l'avons intégré dans un Web sémantique d'entreprise. Ce système nous fournit des fonctions d'aide permettant de réaliser certaines tâches spécifiques telles que la visualisation des changements effectués, la comparaison des versions de l'ontologie (même dans le cas où on ne sait pas quels sont les changements ontologiques effectués), la détection des annotations sémantiques inconsistantes à cause de modifications de l'ontologie, la correction de ces annotations inconsistantes, etc. Toutes ces fonctions du CoSWEM sont expérimentées et validées dans un contexte et avec des données réelles dans le cadre des projets PALETTE et E-WOK_HUB.

Organisation du manuscrit

Le plan de ce manuscrit s'organise en trois parties principales : en partant de la problématique telle qu'elle est traitée dans une étude bibliographique (c.f. Partie I) et en allant vers la description des solutions proposées et développées afin de résoudre les problèmes déclarés (c.f. Partie II) puis en terminant par une expérimentation et une évaluation du système CoSWEM dans le cadre des projets de recherche (c.f. Partie III).

Partie I - État de l'art

Cette partie présente une vue d'ensemble des différents champs de recherche concernés par notre problématique. Le **Chapitre 1** décrit la vision du Web sémantique (i.e. l'architecture, les composants principaux, les langages de représentation, certains outils de support et applications existantes, etc.). Il aborde l'utilisation des technologies du Web sémantique (i.e. l'ontologie, l'annotation sémantique, la recherche intelligente…) en vue de matérialiser une mémoire d'entreprise en un Web sémantique d'entreprise (WSE) permettant de faciliter l'accès, la manipulation et la réutilisation des connaissances organisationnelles. Ce chapitre présente aussi le problème d'évolution et de gestion des changements du Web sémantique en se focalisant sur quelques recherches existantes concernées.

Le **Chapitre 2** se concentre sur la présentation du Web sémantique d'entreprise et de son évolution dans le contexte d'un environnement dynamique et distribué. Nous nous intéressons aux scénarios d'évolution de l'ontologie et de l'annotation sémantique qui sont deux composants importants du WSE. En particulier, nous donnons un aperçu des aspects importants de ces deux composants, et nous positionnons aussi notre travail avec d'autres travaux existants.

Partie II – Vers le système de gestion de l'évolution CoSWEM

La deuxième partie débute par l'introduction du système de gestion de l'évolution CoSWEM dans le **Chapitre 3**. Nous évoquons les besoins qui ont motivé l'élaboration de ce système, son architecture, ses composants principaux, les processus d'évolution de l'ontologie et de l'annotation sémantique dans CoSWEM ainsi que son intégration dans un Web sémantique d'entreprise. Nous décrivons certains outils d'aide qui sont utilisés dans ce système pour effectuer des tâches spécifiques.

Les deux chapitres suivants décrivent en détail l'évolution de l'ontologie et de l'annotation sémantique respectivement. Dans le **Chapitre 4**, nous considérons l'évolution de l'ontologie comme une partie de la gestion de l'ontologie facilitant la modification d'une ontologie en préservant sa consistance. Nous nous intéressons dans ce processus d'évolution aux trois étapes principales : (i) la représentation des changements, (ii) la résolution des changements et (iii) la propagation des changements de l'ontologie.

Le **Chapitre 5** traite de l'évolution de l'annotation sémantique dans le contexte de modification de l'ontologie. Nous étudions les conséquences des changements ontologiques sur les annotations sémantiques qui utilisent les concepts et les propriétés définies dans l'ontologie. En vue de gérer l'évolution des annotations sémantiques, nous présentons une *approche procédurale* et une *approche basée sur des règles* qui sont implémentées dans CoSWEM. Ces deux approches sont destinées respectivement aux deux scénarios d'évolution *avec trace* et *sans trace* de changement de l'ontologie.

Partie III – Exploitation et évaluation du système CoSWEM

La troisième et dernière partie présente l'implémentation et l'expérimentation du système CoSWEM. Dans le **Chapitre 6**, nous décrivons le processus d'implémentation du système CoSWEM. Nous poursuivons par une expérimentation du système sur les projets PALETTE et E-WOK_HUB ainsi que sur un ensemble de données de test. Finalement, nous présentons les résultats des expériences menées en particulier sur les phases de détection et de correction des inconsistances. Nous présentons également une évaluation de ces résultats et mettons en évidence les avantages du système.

Pour conclure les travaux présentés ci-dessus, dans le **Chapitre Conclusion**, nous faisons un résumé de la problématique, de nos propositions et des principales approches suivies. Nous discutons également les limites et les points restant à approfondir dans le futur.

Ce manuscrit comporte également trois annexes :

L'**Annexe-A** présente l'ensemble des stratégies de résolution pour l'ontologie et pour l'annotation sémantique qui correspondent à chaque cas de changement ontologique. Dans l'**Annexe-B**, nous présentons la représentation en RDF(S) de l'ontologie d'évolution qui sert à modéliser les types de changements ainsi que les informations concernant le processus d'évolution. Enfin dans l'**Annexe-C**, nous décrivons un exemple d'une trace d'évolution qui capture des changements effectués entre deux versions de l'ontologie. Cette trace est représentée sous forme d'une annotation en RDF qui repose sur l'ontologie d'évolution.

Partie I

État de l'art

Chapitre 1

État de l'art

Ce chapitre est consacré à une étude bibliographique qui concerne étroitement notre travail de recherche. Puisque le contexte de travail s'intéresse au problème de l'évolution d'un Web sémantique d'entreprise, l'état de l'art présenté dans ce manuscrit va aborder certains domaines participant à nos recherches : Web sémantique, Mémoire d'entreprise (Mémoire organisationnelle) et Gestion des connaissances, Évolution du système de gestion des connaissances. Nous allons d'abord dresser le portrait du Web sémantique en représentant son architecture, ses langages de représentation ainsi que ses composants principaux. Nous présentons ensuite le besoin de matérialiser une Mémoire d'entreprise vers un Web sémantique d'entreprise, qui sera une approche de gestion des connaissances, ainsi que son problème d'évolution dans le nouvel environnement du Web sémantique. Enfin, nous faisons une synthèse de différentes recherches existantes sur l'évolution des composants du système de gestion des connaissances. Nous présentons également les outils et les prototypes développés dans ces recherches récentes.

1.1 Web sémantique

De nos jours, les informations du web actuel augmentent énormément ce qui a bien renforcé le besoin de partager, d'échanger et de réutiliser ces informations d'une manière efficace entre les utilisateurs. Cependant, la plupart des informations se trouvant sur Internet sont effectivement lisibles par les hommes mais elles sont difficilement interprétables par les machines. Nous pouvons le constater clairement à travers un exemple de recherche d'information avec un moteur de recherche actuel basé principalement sur les mots-clés (e.g. Altavista, Google…). Par exemple, si nous effectuons une recherche sur les mots "voiture" et "accident" avec le sens d'un accident de voiture mais nous ne l'avons pas spécifié explicitement au moteur de recherche, alors nous obtiendrons peut-être beaucoup de réponses renvoyées qui ne relient pas du tout avec notre requête. Elles pourraient être des documents contenant le mot "voiture" avec le sens d'un magazine de voiture, d'une nouvelle collection de voiture…, ou les documents comportant le mot "accident" avec le sens d'un accident d'avion, d'un lieu ou des victimes de l'accident, etc. Cependant, les résultats renvoyées peut-être ne contiennent pas des informations telles que "collision des voitures" ou "accident des camions"… qui sont également considérées d'être concernées à notre requête.

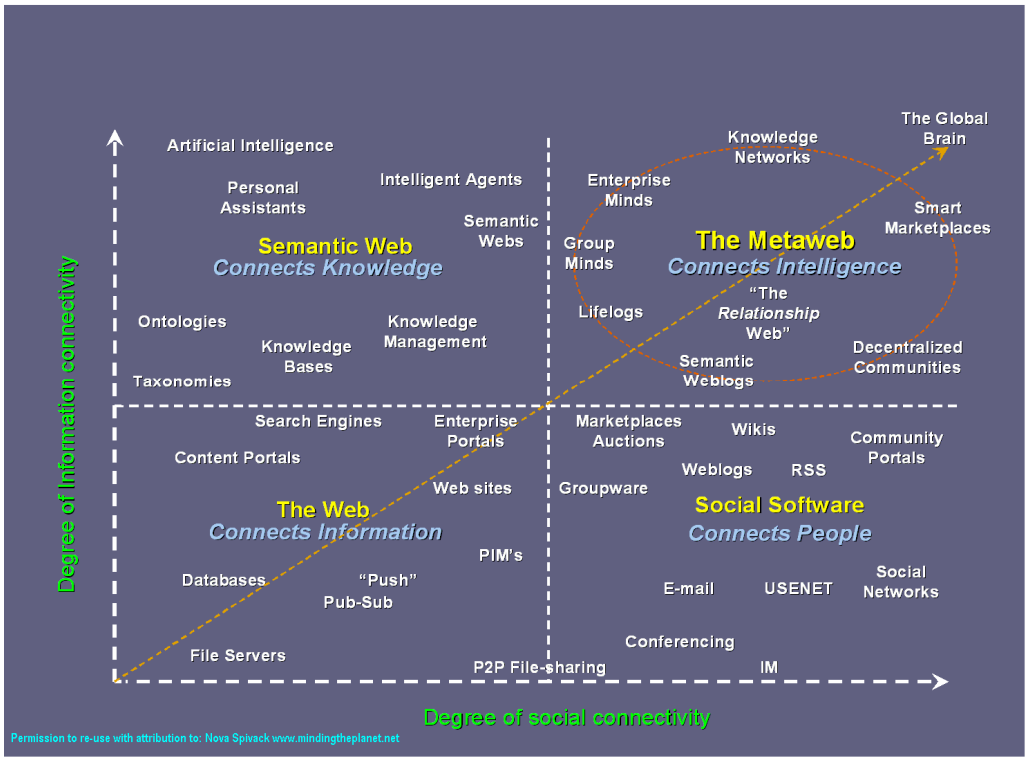


Figure 1 - Le web du futur [Spivack, 2004]

La raison principale de l'imprécision de la recherche ci-dessus est que la description de ces informations est encore limitée et non pas exploitée d'une manière plus "intelligente" par les machines. Le manque de normes et de

sémantique formelle des informations du Web actuel empêche également le partage et la réutilisation d'information entre les individus et les organisations. Par conséquent, la vision du Web du futur a pour objectif de relier des "connaissances" en permettant à des machines d'accéder aux sources d'information et aux services du Web (c.f. Figure 1).

L'expression de Web Sémantique, énoncée par Tim Berners-Lee [Berners-Lee et al., 2001] fait d'abord référence à la vision du Web du futur comme un vaste espace d'échange de ressources supportant la collaboration entre humains et machines en vue d'exploiter plus efficacement de grands volumes d'informations et de services variés disponibles sur le Web. L'objectif du Web sémantique est de rendre explicite le contenu sémantique des ressources dans le Web (documents, pages web, services, etc.). Les machines et les agents logiciels pourraient "comprendre" les contenus décrits dans les ressources et faciliter les tâches de traitement des informations de façon plus automatique et plus efficace. Avec cette idée, le moteur de recherche sémantique peut "savoir" et "trouver" exactement les documents contenant l'information "accident de voiture" grâce aux sémantiques attachées dans ces documents telles que *"l'accident est un type de collision", "l'accident entrîne des dégâts sur le véhicule", "la voiture est un type de véhicule", "le camion est un type de véhicule", etc*.

Les recherches actuellement réalisées dans le domaine du Web Sémantique s'appuient sur un existant riche provenant de différents domaines. Le Web sémantique est non seulement appliqué dans les recherches intelligentes d'information, mais il est aussi étudié dans les recherches sur l'ingénierie des connaissances, les systèmes de représentation des connaissances, le traitement automatique de la langue naturelle, l'apprentissage, les agents intelligents, le raisonnement automatique, etc. Dans la section suivante, nous présentons en détail l'architecture et certains langages de représentations du Web sémantique qui sont largement utilisés.

1.1.1 Architecture et langages du Web sémantique

1.1.1.1 Architecture du Web sémantique

La vision courante du Web sémantique proposée par [Berners-Lee et al., 2001] peut être représentée dans une architecture en plusieurs couches différentes (c.f. Figure 2).

Les couches les plus bases assurent l'interopérabilité syntaxique : la notion d'URI [5] fournit un adressage standard universel permettant d'identifir les ressources tandis que Unicode est un encodage textuel universel pour échanger

[5] Uniform Resource Identifier

des symboles. Rappelons que l'URL[6], comme l'URI, est une chaîne courte de caractères qui est aussi utilisée pour identifier des ressources (physiques) par leur localisation.

XML[7] fournit une syntaxe pour décrire la structure du document, créer et manipuler des instances des documents. Il utilise l'espace de nommage (namespace) afin d'identifier les noms des balises (tags) utilisées dans les documents XML. Le schéma XML permet de définir les vocabulaires pour des documents XML valides. Cependant, XML n'impose aucune contrainte sémantique à la signification de ces documents, l'interopérabilité syntaxique n'est pas suffisante pour qu'un logiciel puisse "comprendre" le contenu des données et les manipuler d'une manière significative.

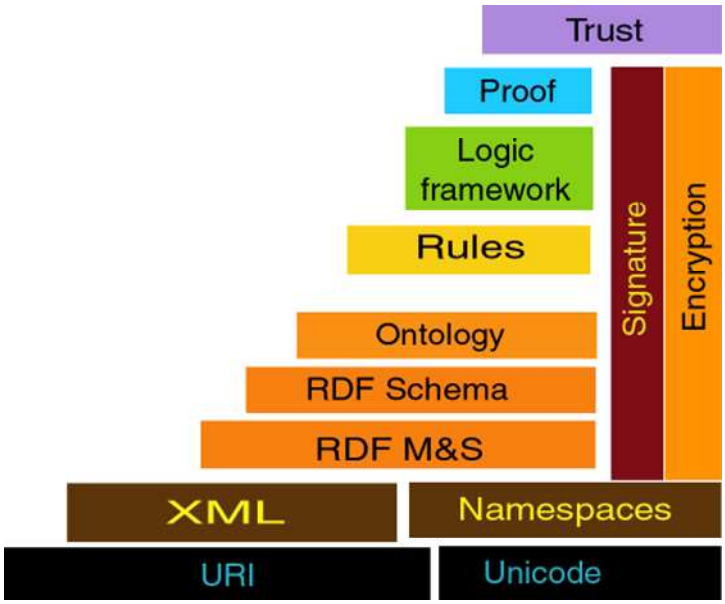


Figure 2 - Architecture du Web sémantique [Berners-Lee et al., 2001]

Les couches RDF M&S (RDF Model and Syntax) et RDF Schéma sont considérées comme les premières fondations de l'interopérabilité sémantique. Elles permettent de décrire les taxonomies des concepts et des propriétés (avec leurs signatures). RDF[8] fournit un moyen d'insérer de la sémantique dans un document, l'information est conservée principalement sous forme de déclarations RDF. Le schéma RDFS décrit les hiérarchies des concepts et des relations entre les concepts, les propriétés et les restrictions domaine/co-domaine pour les propriétés. Les trois langages XML, RDF et RDFS seront présentés plus en détail dans la partie suivante.

[6] Uniform Resource Locator

[7] Extensible Markup Language

[8] Resource Description Framework

La couche suivante Ontologie décrit des sources d'information hétérogènes, distribuées et semi-structurées en définissant le consensus du domaine commun et partagé par plusieurs personnes et communautés. Les ontologies aident la machine et l'humain à communiquer avec concision en utilisant l'échange de sémantique plutôt que de syntaxe seulement.

Les règles sont aussi un élément clé de la vision du Web sémantique, la couche Règles offre la possibilité et les moyens de l'intégration, de la dérivation, et de la transformation de données provenant de sources multiples, etc.

La couche Logique se trouve au-dessus de la couche Ontologie. Certains considèrent ces deux couches comme étant au même niveau, comme des ontologies basées sur la logique et permettant des axiomes logiques. En appliquant la déduction logique, on peut inférer de nouvelles connaissances à partir d'une information explicitement représentée.

Les couches Preuve (Proof) et Confiance (Trust) sont les couches restantes qui fournissent la capacité de vérification des déclarations effectuées dans le Web Sémantique. On s'oriente vers un environnement du Web sémantique fiable et sécurisé dans lequel nous pouvons effectuer des tâches complexes en sûreté. D'autre part, la provenance des connaissances, des données, des ontologies ou des déductions est authentifiée et assurée par des signatures numériques, dans le cas où la sécurité est importante ou le secret nécessaire, le chiffrement est utilisé.

1.1.1.2 Langages du Web sémantique

Dans le contexte du Web Sémantique, plusieurs langages ont été développés. La plupart de ces langages reposent sur XML ou utilisent XML comme syntaxe. Nous allons présenter brièvement certains langages principaux XML, XML Schéma, RDF(S), OWL, SPARQL, RDFa et RIF.

XML (eXtensible Markup Language)

XML[9] est un langage de balisage extensible et est considéré comme une spécification pour les documents "lisible par les machines". Il est naturellement utilisé pour encoder les langages du Web sémantique. Le balisage signifie que certaines suites des caractères du document peuvent contenir de l'information indiquant le rôle du contenu du document. XML se sert de balises (tags, par exemple <nom> ou <adresse>) pour décrire un classement des données du document et sa structure logique. Cependant, le caractère extensible indique la différence importante avec d'autres langages précédents qui est aussi la caractéristique essentielle du XML. XML est un métalangage (une description de type de document, DTD, permet de décrire la grammaire des documents

[9] http://www.w3.org/XML/

admissibles) : en effet XML fournit un structure pour représenter d'autres langages d'une manière normalisée.

Un document XML a toujours un élément racine. Des éléments XML peuvent contenir d'autres éléments qui sont représentés par les balises d'ouverture et de clôture. Examinons un exemple d'un morceau du document XML :

```
1  <?xml version="1.0"?>
2  <etudiant>
3        <nom>Luong Phuc Hiep</nom>
4        <phone>5096</phone>
5        <bureau>
6              <numero>BS04</numero>
7              <batiment>Borel</batiment>
8        </bureau>
9  <etudiant>
```

DTD (Document Type Definition) et Schéma XML

Une DTD permet de définir formellement l'ensemble des éléments, des attributs et des entités qui sont utilisés dans le document XML. La DTD spécifie l'imbrication des éléments, les attributs possibles des documents et les types des attributs. En définissant les types de DTD, on peut aussi vérifier si un document est valide ou pas grâce aux types définis dans la DTD. La partie suivante est un exemple d'une DTD correspondant au morceau de document XML précédent :

```
1  <!DOCTYPE etudiant [
2        <!ELEMENT etudiant (nom, phone,bureau)>
3        <!ELEMENT nom (#PCDATA)>
4        <!ELEMENT phone (#PCDATA)>
5        <!ELEMENT bureau(numero, batiment)>
6        <!ELEMENT numero (#PCDATA)>
7        <!ELEMENT batiment (#PCDATA)>]
8  >
```

XML Schéma[10], une nouvelle recommandation du W3C, est un langage XML qui peut remplacer la DTD. XML schéma présente des types de données XML. Il est possible de définir des types de données complexes en utilisant des éléments imbriqués. XML schéma a plusieurs avantages par rapport à la DTD : il offre tout d'abord une grammaire plus riche pour décrire la structure des éléments. Ensuite, il fournit un mécanisme d'inclusion et de dérivation qui permet de réutiliser les définitions des éléments communs ou bien d'adapter une définition existante à une nouvelle. Enfin, XML schéma utilise l'espace de nommage (namespace) XML qui permet d'identifier une définition du document spécifique avec un nom unique et de préfixer toutes les balises avec ce nom unique. L'exemple du schéma XML suivant est équivalent à la DTD au-dessus :

[10] http://www.w3.org/XML/Schema

```
1  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
2  <xs:element name="etudiant">
3          <xs:complexType>
4                  <xs:sequence>
5                  <xs:element name="nom" type="xs:string"/>
6                  <xs:element name="phone" type="xs:string"/>
7                  <xs:element name="bureau" type="xs:string"/>
8                  <xs:complexType>
9                          <xs:sequence>
10                         <xs:element name="numero" type="xs:string"/>
11                         <xs:element name="batiment" type="xs:string"/>
12                         </xs:sequence>
13                 </xs:complexType>
14                 </xs:sequence>
15         </xs:complexType>
16 </xs:element>
17 </xs:schema>
```

RDF (Resource Description Framework) - modèle de donnée RDF

RDF est un formalisme pour la description des méta-données; il fournit l'interopérabilité entre les applications qui échangent des informations sur le Web qui peuvent être compréhensible par les machines [Lassila and Swick, 1999]. RDF augmente la facilité de traitement automatique des ressources Web. Il peut être utilisé pour annoter des documents écrits dans des langages non structurés, ou comme une interface pour des documents écrits dans des langages ayant une sémantique équivalente. La syntaxe de RDF repose sur le langage de balisage extensible XML. XML fournit une syntaxe pour encoder des données tandis que RDF fournit un mécanisme décrivant le sens des données. Un des buts de RDF est de rendre possible la spécification de la sémantique des données basées sur XML d'une manière standardisée et interopérable.

Un document RDF est un ensemble de triplets de la forme `<ressource, propriété, valeur>`. Une ressource est une entité accessible via un URI[11] sur l'Internet (par exemple un document HTML ou XML, une image, une page Web ou même une partie de la page Web). Une propriété définit une relation binaire entre une ressource et une valeur (associer de l'information à une ressource). Une valeur est une ressource ou une valeur littérale (chaîne de caractères). Un énoncé (ou une déclaration) RDF spécifie la valeur d'une propriété d'une ressource. On peut le décrire comme `propriété(ressource, valeur)`. Les éléments de ces triplets peuvent être des URIs, des littéraux ou des variables. Cet ensemble de triplets peut être représenté de façon naturelle par un graphe (plus précisément un multi-graphe orienté étiqueté), où les éléments apparaissant comme ressource ou valeur sont les sommets, et chaque triplet est représenté par un arc dont l'origine est sa ressource et la destination sa valeur comme la montre la figure suivante (c.f. Figure 3).

[11] Uniform Resource Identifier

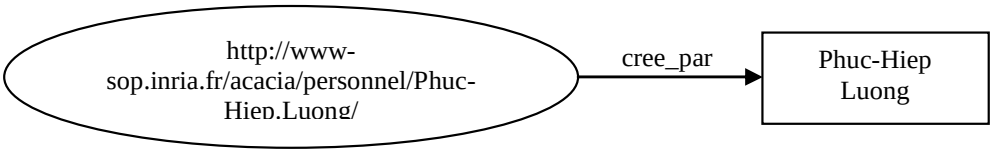


Figure 3 - Modèle de triplet en RDF

En utilisant ce modèle de triplet, le fait *"La page Web à l'adresse http://www-sop.inria.fr/acacia/personnel/Phuc-Hiep.Luong/ est créée par Phuc-Hiep Luong"* est présenté en RDF comme suit :

```
1   <?xml version="1.0"?>
2   <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3   xmlns:dc="http://purl.org/dc/elements/1.1/">
4   <rdf:Description rdf:about="http://www-sop.inria.fr/acacia/personnel/Phuc-Hiep.Luong/">
5         <cree_par>
6               <rdf:Description>
7                     <dc:nom>Luong Phuc Hiep</dc:nom>
8               </rdf:Description>
9         </cree_par>
10  </rdf:Description>
11  </rdf:RDF>
```

RDFS (Resource Description Framework Schema) – méta-modèle

Le langage RDFS a été développé en se basant sur RDF [Brickley and Guha, 2000]. Le modèle des données RDF ne précise que le mode de description des données mais ne fournit pas la déclaration des propriétés spécifiques au domaine ni la manière de définir ces propriétés avec d'autres ressources. RDFS a pour but d'étendre RDF en décrivant plus précisément les ressources utilisées pour étiqueter les graphes. Pour cela, il fournit un mécanisme permettant de spécifier les classes dont les instances seront des ressources, comme les propriétés.

RDFS s'écrit toujours à l'aide de triplets RDF en utilisant deux propriétés fondamentales `subClassOf` et `type` pour représenter respectivement les relations de subsomption entre classes et les relations d'instanciation entre instances et classes. Les classes spécifiques au domaine sont déclarées comme des instances de la ressource `Class` et les propriétés spécifiques au domaine comme des instances de la ressource `Property`. Les propriétés `subClassOf` et `subPropertyOf` permettent de définir des hiérarchies de classes et de propriétés. D'autre part, RDFS ajoute à RDF la possibilité de définir les contraintes de domaine et co-domaine de valeurs avec l'aide des attributs `rdfs:domain` et `rdfs:range`. Les ressources instances sont décrites en utilisant le vocabulaire donné par les classes définies dans ce schéma. Pour résumer, XML peut être vu comme la couche de transport syntaxique, RDF comme un langage relationnel de base. RDFS offre des primitives de représentation de structures ou primitives ontologiques [Laublet et al., 2002].

```
1   <rdf:RDF>
2   …
3   <rdfs:Class rdf:ID="Personne">
4         <rdfs:subClassOf rdf:resource="#Entité"/>
5   </rdfs:Class>
6   <rdfs:Class rdf:ID="Etudiant">
7         <rdfs:subClassOf rdf:resource="#Personne"/>
8   </rdfs:Class>
9   …
10  <rdf:Property rdf:ID="travailler_dans">
11        <rdfs:domain rdf:resource="#Personne"/>
12        <rdfs:range rdf:resource="#Institut"/>
13  </rdf:Property>
14  …
15  </rdf:RDF>
```

OWL (Web Ontology Language)

OWL [12] est un langage pour représenter des ontologies dans le Web sémantique. C'est une extension du vocabulaire de RDF(S). Le langage OWL offre aux machines de plus grandes capacités d'interprétation du contenu Web que celles permises par XML, RDF et RDF schéma (RDFS), grâce à un vocabulaire supplémentaire et une sémantique formelle. Inspiré des logiques de descriptions (et successeur de DAML+OIL), OWL fournit un grand nombre de constructeurs permettant d'exprimer de façon très fine les classes de manière plus complexe correspondant aux connecteurs de la logique de description équivalente (intersection, union, restrictions diverses, etc.), les propriétés des classes définies (telles que la disjonction), la cardinalité (par exemple "exactement un"), plus des types des propriétés (propriétés d'objet ou d'annotation…), des caractéristiques des propriétés (par exemple la symétrie, la transitivité), et des classes énumérées.

Le langage OWL se compose de trois sous-langages offrant une expressivité croissante [Bechhofer et al., 2003]: OWL-Lite, OWL-DL et OWL-Full.

- OWL-Lite : ce sous-langage ne contient qu'un sous-ensemble réduit des constructeurs disponibles. Il a la complexité formelle la plus basse et l'expressivité minimale dans la famille OWL. Il est suffisant pour représenter des thésaurus et d'autres taxonomies ou des hiérarchies de classification avec des contraintes simples.

- OWL-DL contient l'ensemble des constructeurs, mais avec des contraintes particulières sur leur utilisation qui assurent la décidabilité de la comparaison de types. Par contre, la grande complexité de ce langage semble rendre nécessaire une approche heuristique.

- OWL-Full : Ce sous-langage est conçu pour ceux qui ont besoin de l'expressivité maximale, de la liberté syntaxique de RDF mais sans garantie de calculabilité.

[12] http://www.w3.org/2004/OWL/

Voici un exemple d'une ontologie en OWL :

```
1.  <rdf:RDF
2.  xmlns:owl ="http://www.w3.org/2002/07/owl#"
3.  xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4.  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
5.  xmlns:xsd ="http://www.w3.org/2001/XMLSchema#"
6.  xmlns     ="http://www.inria.fr/acacia/exemple/animals.owl#"
7.  xml:base  ="http://www.inria.fr/acacia/exemple/animals.owl#" >

8.  <owl:Class rdf:ID="Animal">
9.      <rdfs:label>Animal</rdfs:label>
10. </owl:Class>
11. <owl:Class rdf:ID="Person">
12.     <rdfs:subClassOf rdf:resource="#Animal"/>
13.     <rdfs:subClassOf>
14.         <owl:Restriction>
15.             <owl:onProperty rdf:resource="#hasParent"/>
16.             <owl:allValuesFrom rdf:resource="#Person"/>
17.         </owl:Restriction>
18.     </rdfs:subClassOf>
19. </owl:Class>
20. <owl:ObjectProperty rdf:ID="hasAncestor">
21.     <rdf:type rdf:resource="&owl;TransitiveProperty"/>
22.     <rdfs:domain rdf:resource="#Animal"/>
23.     <rdfs:range rdf:resource="#Animal"/>
24. </owl:ObjectProperty>
25.  <owl:ObjectProperty rdf:ID="hasParent">
26.     dfs:subPropertyOf rdf:resource="#hasAncestor"/>
27. </owl:ObjectProperty>
28. </rdf:RDF>
```

SPARQL (Query Language for RDF)

La représentation de connaissances dans la nouvelle génération du Web sémantique est importante mais la capacité de faire des requêtes de ces connaissances joue aussi un rôle crucial dans les applications ou les bases de connaissances. Parmi certains langages de requêtes connus tels que RQL, TRIPLE, SQL, XPATH… SPARQL, un langage de requête proposé par W3C et dédié à RDF, est largement utilisé dans le domaine de recherche et d'extraction d'informations [Prud'hommeaux et Seaborne, 2007]. Le langage SPARQL est basé sur la correspondance des patrons de graphe (matching graph patterns). Le patron de graphe le plus simple est le patron de triplets (comme un triplet en RDF) mais il possède la capacité d'exprimer des variables de requête dans les positions du sujet, de la propriété ou de l'objet d'un triplet. D'autre part, SPARQL intègre également des balises (tags) spécifiques telles que le patron de graphe optionnel, l'union et l'intersection des patrons, le filtrage, les opérateurs de comparaison des valeurs... permettant d'effectuer des requêtes plus efficaces et flexibles. Nous présentons par la suite un exemple de requête en SPARQL qui demande le titre (i.e. ?title) et le prix (i.e. ?price) des ressources dont le prix est inférieur à 30.

```
1.  PREFIX dc: <http://purl.org/dc/elements/1.1/>
2.  PREFIX ns: <http://example.org/ns#>
3.   SELECT ?title ?price
4.   WHERE { ?x ns:price ?price .
5.           ?x dc:title ?title .
             FILTER (?price < 30) .
6.  }
```

Dans notre travail de thèse, nous utilisons le moteur de recherche sémantique Corese dont le langage de requête est aussi basé sur le langage SPARQL [Corby et Faron, 2007]. La syntaxe et la sémantique du langage de requête de Corese respectent les protocoles de SPARQL.

RDFa

RDFa [Adida et Birbeck, 2007] est un langage candidat à une recommandation du W3C. Son objectif principal est de rajouter les informations en RDF dans les documents HTML ou XHTML (i.e. (X)HTML). Le langage RDFa fournit une syntaxe et un ensemble de balises (tags) pour décrire les données structurées en (X)HTML. Avec les informations supplémentaires insérées dans les balises, les données structurées en (X)HTML sont rajoutées les "sémantiques" qui permettront l'échange d'informations par les applications automatiques ou par les agents informatiques. L'exemple ci-dessous représente les informations ajoutées par RDFa (i.e. le namespace xmlns:contact, les propriétés contact:fn, contact:title, etc.) dans un document XHTML décrivant les informations d'un coordonnée (i.e. le nom, le titre et l'adresse email du contact).

```
1.  <html  xmlns:contact="http://www.w3.org/2001/vcard-rdf/3.0#">
2.  ...
3.  <p class="contactinfo" about="http://example.org/staff/Hiep">
4.  Je m'appelle
5.  <span property="contact:fn">
6.   Luong Phuc Hiep
7.  </span>
8.  I'm a
9.  <span property="contact:title">
10. Doctorant
11. </span>
12. Adresse email
13. <a rel="contact:email" href="mailto:phluong@sophia.inria.fr">
14. via email
15. </a>.
16. </p>
17. ...
```

RIF (Rule Interchange Format)

RIF [Boley et Kifer, 2007] est un formalisme proposé par le W3C permettant de fournir l'interopérabilité entre les langages de règles en général et ceux utilisés en particulier pour le Web. Le noyau de ce langage, RIF Core, correspond à la logique de Horn. Il offre certaines extensions inspirées des langages à objets et de frames, ainsi que les URIs, les types de données de XML Schema… La partie principale de RIF Core est le langage de condition (Condition Language). Ce langage définit la syntaxe et la sémantique des règles de RIF ainsi

que la syntaxe pour des requêtes. D'autre part, la spécification de RIF repose sur certains types de règles : les règles de production, la programmation logique, les règles basées sur la logique du premier ordre, les règles réactives ou les règles normatives… Nous montrons ci-dessous une condition représentée en RIF et la correspondance en règle RIF Horn.

1. **RIF condition:**
   Exists ?Y (condition(?X ?Y))

2. **RIF Horn rule:**
   Forall ?X (then(?X) :- Exists ?Y (condition(?X ?Y)))

1.1.2 Composants principaux du Web sémantique

Nous examinons dans cette section les deux composants essentiels et importants du Web sémantique que sont l'ontologie et l'annotation sémantique. Les ontologies sont la technologie dorsale pour le Web sémantique et - plus généralement - pour le management des connaissances formalisées décrivant les ressources du Web. Elles fournissent la "sémantique" exploitable par machine des données et des sources d'informations qui peuvent être communiquées entre différents agents (logiciel et humaines), tandis que les annotations sémantiques décrivent les ressources en utilisant la "sémantique" définie dans l'ontologie. Les ressources annotées par les méta-données faciliteront la recherche, l'extraction, l'interprétation et le traitement de l'information d'une manière plus efficace.

1.1.2.1 Ontologies

L'importance des ontologies est reconnue dans divers domaines de recherche comme la représentation des connaissances, l'ingénierie des connaissances, la conception de bases de données, l'intégration d'information, les systèmes d'information… Les ontologies sont aussi centrales pour le Web sémantique qui, d'une part, cherche à s'appuyer sur des modélisations de ressources du Web à partir de représentations conceptuelles des domaines concernés et, d'autre part, a pour objectif de permettre à des programmes de faire des inférences dessus. Nous allons présenter par la suite les définitions les plus citées et la classification de l'ontologie ainsi que certains outils de construction de l'ontologie qui sont largement utilisés d'aujourd'hui.

Définitions et classification

Le terme *ontologie* est initialement emprunté la philosophie signifiant "explication systématique de l'existence". Une ontologie est similaire à un dictionnaire ou un glossaire mais avec une structure détaillée et grande qui permet aux machines de traiter son contenu. Le mot *ontologie* semble susciter beaucoup

de débats ainsi que de définitions dans la communauté de l'Intelligence Artificielle.

Une des définitions les plus célèbres et la plus utilisée est celle de [Gruber, 1993] : "une ontologie est une *spécification explicite* d'une *conceptualisation*". Une *conceptualisation* est une abstraction du monde que nous souhaitons représenter dans un certain but. La conceptualisation est le résultat d'une analyse ontologique du domaine étudié. L'ontologie est une *spécification* parce qu'elle représente la conceptualisation dans une forme concrète. Elle est *explicite* parce que tous les concepts et les contraintes utilisés sont explicitement définis. Une ontologie exprime la conceptualisation explicitement dans un langage formel. Une définition explicite et formelle permet aux agents de raisonner et d'inférer de nouvelles connaissances.

[Studer et al., 1998] a raffiné la définition ci-dessus comme une "*spécification formelle* et *explicite* d'une *conceptualisation partagée*". Une ontologie doit être lisible par la machine en se représentant *formelle* des termes ou des vocabulaires définis. En plus, l'ontologie représente un consensus accepté par une communauté d'utilisateurs, elle est partagée et n'est pas la propriété d'un individu.

Avec l'aide d'une interprétation en termes d'un modèle de l'ontologie, les connaissances du domaine sont représentées par une sémantique formelle. Les connaissances du domaine traduites par une ontologie sont véhiculées normalement par les éléments principaux suivants :

- *Concepts (ou classes) :* représentent les objets, abstraits ou concrets, élémentaires ou composites, du monde réel. Les concepts sont organisés en taxonomie par l'utilisation de la relation de subsomption. Par exemple, les concepts `Personne` et `Doctorant` représentent des classes différentes des objets humains en réalité dans lesquelles `Personne` est un super-concept de `Doctorant`, le concept `Manuscrit` décrit une classe des objets du type de document, etc.

- *Relations :* représentent des liens sémantiques de la connaissance du domaine. Une relation peut être distinguée comme une propriété ou un attribut. Les propriétés décrivent des interactions entre concepts, elles peuvent être fortement typées, i.e., associées un domaine et un co-domaine précis qui permettent respectivement de spécifier les classes susceptibles d'initialiser une propriété et de contraindre les domaines de valeurs des propriétés. Dans l'exemple ci-dessus, une propriété *`rédiger`* peut indiquer une relation entre les concepts `Doctorant` et `Manuscrit` dans laquelle `Doctorant` est le domaine et `Manuscrit` est le co-domaine de la propriété *`rédige`*r. Les attributs correspondent à des caractéristiques, des spécificités particulières, attachées à un concept et qui permettent de le définir de manière unique dans le domaine. Leurs valeurs sont littérales, i.e. de type primitif, comme une chaîne de

caractères ou un nombre entier. Par exemple, le concept `Personne` peut avoir les attributs tels que : avoir un *`nom`*, une *`date de naissance`*, une *`adresse`*, etc.

- *Axiomes :* constituent des assertions (ou des prédicats) acceptées comme vraies qui s'appliquent sur les classes ou les instances des classes de l'ontologie et qui permettent de restreindre les interprétations possibles d'une ontologie et/ou de déduire de nouveaux faits à partir des faits connus.

- *Instances (ou objets ou individus) :* représentent des éléments spécifiques d'un concept (ou d'une classe). Les instances peuvent être associées à un identifiant unique qui permet de distinguer une instance à une autre. Par exemple, l'instance "Phuc-Hiep" est du type concept `Doctorant`, "Edelweiss" est une instance du concept `Equipe_Recherche`, etc.

La Figure 4 présente un extrait d'une ontologie décrivant les connaissances du domaine de la recherche scientifique, par exemple des activités d'un institut de recherche. Dans cet extrait, nous avons des concepts tels que `Personne`, `Doctorant`, `Chercheur` qui sont classés en ordre hiérarchique… pour représenter des employés qui travaillent dans (i.e. la propriété) les équipe de recherche (i.e. le concept `Equipe_Recherche`) ou dans le département d'administration (i.e. le concept `Administration`), etc.

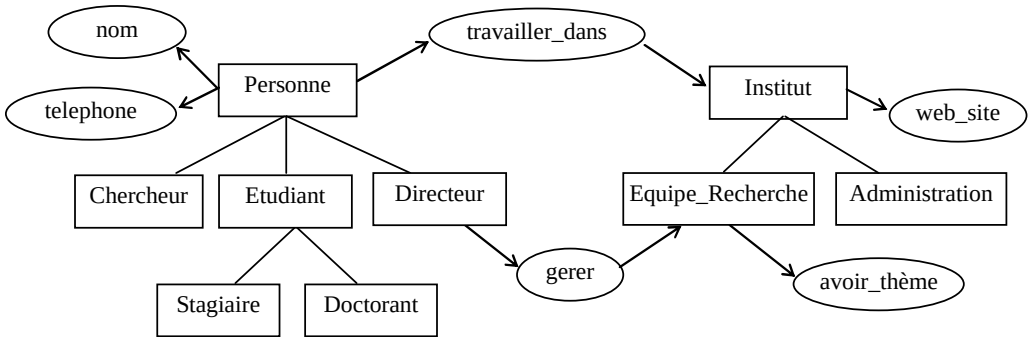


Figure 4 - Exemple d'un extrait de l'ontologie

Pour la typologie des ontologies, on peut distinguer différents niveaux d'ontologies selon le but pour lequel elles sont conçues. La classification de [Van Heijst et al., 1997] repose sur deux critères : le sujet et la structure d'une conceptualisation. En ce qui concerne le sujet de la conceptualisation, les auteurs distinguent :

- Les ontologies de domaine : les plus connues, elles expriment des conceptualisations spécifiques à un domaine, elles sont réutilisables pour des applications sur ce domaine.

- Les ontologies d'application : elles contiennent des connaissances du domaine nécessaires à une application donnée ; elles sont spécifiques et non réutilisables.

- Les ontologies génériques : appelées aussi ontologies de haut niveau, elles expriment des conceptualisations très générales tels que le temps, l'espace, l'état, le processus, les composants, elles sont valables dans différents domaines; les concepts figurant dans une ontologie du domaine sont subsumés par les concepts d'une ontologie générique, la frontière entre les deux étant floue.

- Les ontologies de représentation ou méta-ontologies : indiquent des formalismes de représentation de la connaissance ; les ontologies génériques ou du domaine peuvent être écrites en utilisant des primitives d'une telle ontologie.

D'autre part, les ontologies sont aussi classées en plusieurs niveaux d'expression [Uschold et Gruninger, 1996] selon leur utilisation:

- Très informelle : l'ontologie est exprimée en langage naturel.

- Semi-informelle : l'ontologie est exprimé sous une forme restreinte et structurée de langage naturel pour augmenter la clarté et pour réduire l'ambiguïté.

- Semi-formelle : l'ontologie est exprimée en langage formel.

- Rigoureuse formelle : l'ontologie est défini avec une sémantique formelle, permettant le théorème et la preuve.

Méthodologies et Outils de développement des ontologies

La plupart des approches existantes décrivent la construction d'une ontologie à partir de zéro, par exemple les méthodologies les plus souvent citées sont : la méthodologie de [Uschold et King, 1995] est élaborée à la suite de l'expérience de la construction de l'Enterprise Ontology, la méthodologie de [Gruninger et Fox, 1995] est basée sur l'expérience de la construction d'une ontologie dans le cadre du projet Tove, la Methontology de [Fernandez et al., 1997] qui couvre tout le cycle de vie d'une ontologie, La méthode On-To-Knowledge [Sure et Studer, 2002], a été mise en œuvre pour répondre à la problématique du développement d'ontologies dans le cadre du Web sémantique, etc. [Rogozan, 2004] résume certaines approches décrivant la construction d'une ontologie par fusion d'ontologies différentes, par exemple les méthodes FCA-Merge et PROMPT ou la méthodologie MOMIS. D'autres méthodologies se focalisent sur une des étapes du processus de représentation des connaissances, par exemple la méthodologie présentée dans [Aussenac-Gilles et al., 2000] insistant sur l'approche basée sur l'analyse d'un corpus textuel. La méthodologie

OntoSpec de [Kassel, 2002] constitue une aide à la structuration des hiérarchies de concepts et de relations durant la phase d'ontologisation.

Il existe aussi plusieurs outils pour développer et maintenir des ontologies. [Gomez-Perez et al., 2002] a fait un résumé de nombreux outils de construction d'ontologies qui ont été développés ces dernières années. Nous introduisons par la suite certains de ces outils qui sont largement utilisés par la communauté des ontologistes.

OILEd[13] est un éditeur d'ontologie graphique développé par l'Université de Manchester qui permet à l'utilisateur de construire des ontologies en langage DAML+OIL. Le modèle de connaissances de l'OILEd est basé sur DAML+OIL mais il supporte aussi un paradigme des formalismes *frames* pour modéliser les connaissances [Bechhofer et al., 2001]. Les classes sont définies en termes de leurs super-classes et les restrictions de propriété avec des axiomes additionnels. Un aspect principal de l'outil OILEd est l'utilisation du raisonneur FaCT pour classifier des ontologies et vérifier leur consistance par la traduction de DAML+OIL à la logique de description en SHIQ.

OntoEdit[14] est un projet développé à l'Université de Karlsruhe qui fournit les méthodologies et processus d'intégration d'ontologie ainsi que l'éditeur d'ontologie [Sure et al., 2002]. Cet outil permet à l'utilisateur de modifier la hiérarchie de concepts (ou de classes), il implémente aussi les interfaces graphiques facilitant le processus de construction de l'ontologie (e.g. les fonctions de copie/colle, de réorganisation de la hiérarchie…). OntoEdit supporte également une interface de *plugs-in* qui permet de rajouter facilement les fonctions nécessaires.

Protégé[15] [Noy et al., 2000] est un outil d'édition de l'ontologie développé à l'Université de Stanford, qui est utilisé largement aujourd'hui pour élaborer des ontologies en RDF(S) et OWL. Protégé fournit un environnement de développement graphique et interactif pour aider les ingénieurs et les experts du domaine à réaliser des tâches plus facilement. Le modèle de connaissances de Protégé est compatible avec l'OKBC (Open Knowledge Base Connectivity). Un des avantages de Protégé est son architecture ouverte et modulaire, il facilite le développement de nouvelles fonctionnalités à travers des *plugs-in* pour effectuer des opérations différentes.

WebODE [16] , développé à l'Université technique de Madrid, est un *benchmark* supportant l'ingénierie de l'ontologie. Il est construit sur un serveur d'application, qui fournit une haute extensibilité et rentabilité en permettant

[13] http://oiled.man.ac.uk/

[14] http://www.ontoknowledge.org/tools/ontoedit.shtml

[15] http://protege.stanford.edu/

[16] http://kw.dia.fi.upm.es/wpbs/

d'ajouter facilement de nouveaux services et fonctionnalités. Les ontologies de WebODE sont représentées en utilisant un modèle de connaissances expressif basé sur la méthodologie de Methontology [Corcho et al., 2002]. Ce modèle contient des concepts (avec des instances et des attributs du concept) des relations, des axiomes et des règles…

1.1.2.2 Méta-données et Annotations sémantiques

L'information du Web actuel devient plus fortement distribuée, extrêmement volumineuse, évolutive, très hétérogène et souvent très peu structurée. Afin de mieux utiliser l'information et les ressources du Web, il est nécessaire de proposer des méthodes et des outils pour représenter, manipuler et exploiter des ressources. Nous allons discuter dans cette section certains moyens et outils qui visent à améliorer la communication et l'interopérabilité des applications ainsi qu'à partager et exploiter l'information sur le Web.

Annotation, méta-données et annotation sémantique

Nous avons peut-être connu les types d'annotation qui sont utilisés comme support de lecture (commentaires, recherche de passages, surlignée, cartes de navigation) ou comme moyen d'explication résumée d'un document. Une ***annotation*** peut être considérée d'une manière simple comme une information graphique ou textuelle attachée à un document et le plus souvent placée dans ce document. En ce qui concerne le Web, les annotations les plus courantes sont des annotations en langage naturel (informelles), des symboliques (surlignée, soulignage, italique), des notes textuelles en marge, des images, des sons, etc.

Normalement, une annotation est toujours associée à l'objet qui a été annoté. Dans ce sens, les annotations sont considérées comme des méta-données. Les ***méta-données*** peuvent être définies comme étant des données relatives à d'autres données: données sur des données. La méta-donnée est une information interprétable par machine sur des ressources d'information du Web ou d'autres sources de données. D'après [Handschuh, 2005], si une méta-donnée est une donnée sur une donnée, une annotation constitue un cas particulier d'une méta-donnée puisqu'elle représente une nouvelle donnée attachée à une ressource documentaire. D'un point de vue plus lié à la pratique de l'annotation et de méta-données, les auteurs de [Charlet et al., 2003] font un distinguo entre ces deux termes :

- *une méta-donnée* sera plutôt attachée à une ressource identifiée en tant que telle sur le Web – aura plutôt une pertinence a priori et sera plutôt saisie suivant un schéma. Par exemple, la description normalisée d'un service Web, l'auteur d'un document, qui permettront de mettre en place des inférences.

- *une annotation* sera plus située au sein de cette ressource et écrite au cours d'un processus d'annotation ou de lecture. Par exemple, un commentaire libre associé à un fragment d'une page Web – quelques mots, un paragraphe – déterminé au besoin.

Puisque les données actuelles du Web sont destinées essentiellement aux humains, elles ne sont pas très bien structurées et n'ont pas de sémantique formelle. Par conséquent, un des objectifs, dans l'environnement du Web Sémantique, est de décrire le contenu des ressources en les annotant avec des informations non ambigües afin de favoriser l'exploitation de ces ressources par des agents logiciels [Prié et Garlatti, 2004]. Cet objectif est considéré comme la tâche d'annotation consistant donc à prendre en entrée une ressource documentaire et fournir en sortie le même contenu enrichi par des annotations sémantiques basées sur des représentations de la connaissance plus ou moins formelles [Amardeilh, 2007].

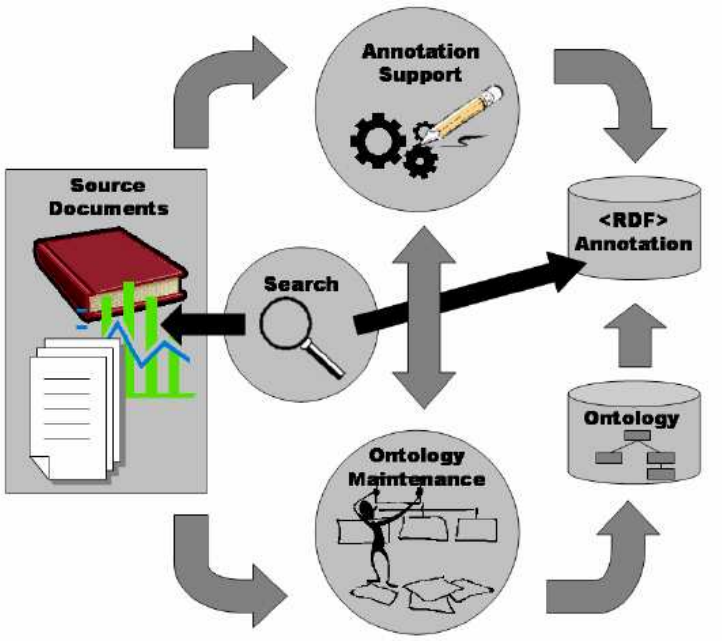


Figure 5 – Rôle des annotation dans le système de gestion des connaissances

La Figure 5 représente le rôle de l'annotation dans un système de gestion des connaissances [Uren et al., 2006]. Les annotations, créées et manipulées par les outils d'annotation, fournissent l'interopérabilité entre différents types de documents et supportent des services de recherche. Les outils de recherche sémantique sont utilisés pour connecter et exploiter les informations attachées dans les annotations et les documents. Les outils de maintenance de l'ontologie supportent le processus d'évolution et de maintenance des modèles de connaissances pour répondre aux changements.

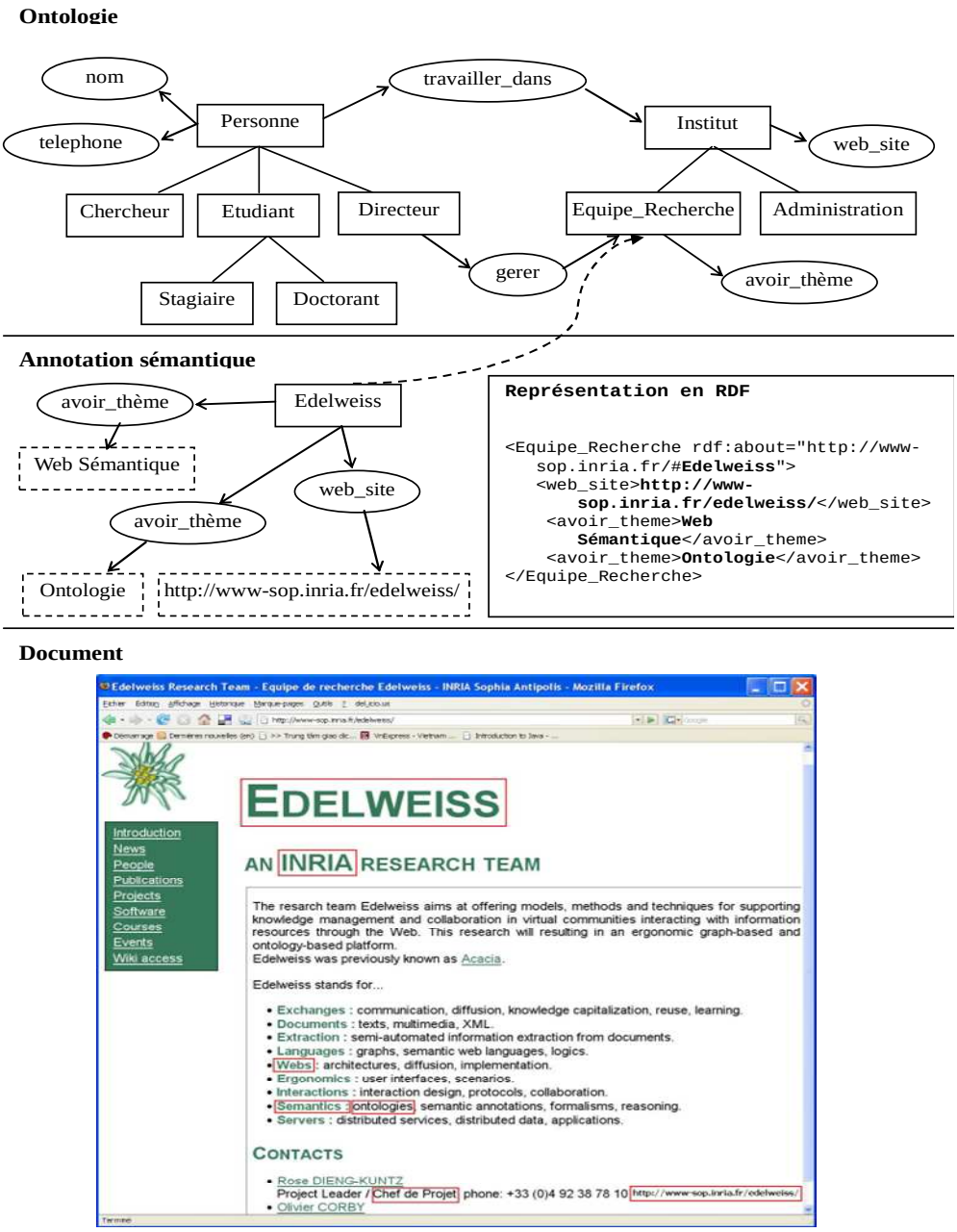


Figure 6 - Utilisation de l'annotation sémantique

Dans le cadre du Web Sémantique, une annotation descriptive, notamment lorsqu'elle s'intéresse à la structure logique du contenu d'un document, est le plus souvent appelée ***annotation sémantique*** [Prié et Garlatti, 2004]. Le terme "sémantique" indique une volonté de faire émerger le sens d'un contenu et ce, de manière plus ou moins formelle selon les préceptes de la logique. Les annotations sémantiques ont donc pour objectif d'exprimer la "sémantique" du contenu d'une ressource. [Amardeilh, 2007] définit l'annotation sémantique comme "une représentation formelle d'un contenu, exprimée à l'aide de concepts, relations et instances décrits dans une ontologie, et reliée à la ressource documentaire source". En termes de documentation, les annotations sémantiques

décrivent le lien entre les entités se trouvant dans le document et leurs descriptions sémantiques représentées dans l'ontologie. Elles permettent ainsi de désambiguïser le contenu du document pour un traitement automatique (ex. recherche documentaire, résumé…). L'annotation sémantique à partir d'ontologie semble actuellement l'approche la plus prometteuse pour partager et exploiter l'information sur le Web.

La Figure 6 donne une vision de l'utilisation de l'annotation sémantique sur le Web sémantique pour annoter un document (i.e. la page Web de l'équipe Edelweiss) en utilisant le vocabulaire défini dans l'ontologie. Les informations de cette page Web telles que le nom de l'équipe, le site web de l'équipe ainsi que certains thèmes de recherche de l'équipe sont annotées à partir de connaissances disponibles dans un extrait de l'ontologie présenté précédemment (c.f. Figure 4). Les annotations sont ensuite regroupées en entrepôts de méta-données. Elles deviennent utiles pour des agents de recherche d'information, faisant ou non appel à des moteurs d'inférence permettant de déduire de nouvelles connaissances formelles des annotations.

Framework et outils de support pour les annotations sémantiques

Pour le processus de création des annotations sémantiques, [Uren et al., 2006] a spécifié certaines exigences qu'un système d'annotations doit satisfaire. Les auteurs ont ensuite résumé les deux *frameworks* principaux des annotations sémantiques qui sont implémentés d'une manière différente dans plusieurs outils d'annotation. Ces deux cadres de travail sont Annotea[17] et CREAM qui est développé à l'université de Karlsruhe.

Annotea [Kahan et al., 2001] [Koivunen, 2005] est un projet du W3C[18] qui spécifie l'infrastructure pour l'annotation des documents du Web. Le format principal utilisé dans l'Annotea est RDF et les types de documents qui peuvent être annotés sont les documents en HTML ou basés sur XML. Annotea fournit dans XPointer une méthode permettant de localiser des annotations dans un document. XPointer est une recommandation du W3C pour identifier des fragments des ressources avec leur URI. L'approche d'Annotea se concentre sur un modèle semi-formel d'annotation dans lequel les annotations contiennent des déclarations qui ont besoin d'être ajoutées à l'aide de méta-données (i.e. l'auteur, la date…)

Le *framework* **CREAM** [Handschuh et al., 2003], comme Annotea, souscrit également aux formats standards du W3C avec des annotations représentées en RDF ou OWL et au XPointer. Ces annotations sont utilisées pour annoter des ressources et des documents dont les formats sont en XML et HTML. Il spécifie

[17] http://www.w3.org/2001/Annotea/

[18] http://www.w3.org/

les composants exigés pour un système d'annotation qui doit supporter la création (semi-) automatique d'annotation basée sur l'ontologie.

Basées sur les deux *frameworks* d'annotation principaux ci-dessus, plusieurs recherches et outils ont été proposés afin de manipuler les annotations au cours du développement du Web Sémantique. [Gomez-Perez et al., 2002] et [Uren et al., 2006] ont fait un très bon état de l'art sur les outils d'annotations existants. Ces outils peuvent supporter le processus d'évolution d'une manière automatique, semi-automatique et manuelle. Nous présentons dans cette section quelques outils, parmi les plus connus, qui nous permettent d'élaborer une base d'annotations à partir des ressources et des documents de texte.

MnM est un outil d'annotation développé dans un projet de AKT[19] [Vargas-Vera et al., 2002]. Il fournit un environnement pour la génération semi-automatique d'annotations sémantiques associées aux documents Web en utilisant une ontologie existante. Cet outil est basé essentiellement sur des techniques d'apprentissage et des méthodes de TALN[20]. Il permet l'annotation d'un corpus d'apprentissage en se basant sur l'ontologie du domaine, ensuite ce corpus est traité dans l'outil Amilcare [Ciravegna, 2003] pour générer un ensemble de règles d'extraction qui seront utilisées pour le futur usage. Dans le processus d'annotation, MnM utilise deux types de règles : (i) les règles d'étiquetage qui permettent de repérer la partie du texte à annoter et d'y insérer une étiquette sémantique et (ii) les règles de correction qui permettent d'explorer le texte étiqueté et de détecter les étiquettes incorrectes et de les corriger tout en se basant sur les informations recueillies lors de la phase d'apprentissage.

KIM [Popov et al., 2004] utilise des techniques d'extraction d'information pour élaborer une base d'annotations. Cet outil fournit une plate-forme de génération automatique d'annotations sémantiques et de recherche documentaire basée sur ces annotations. L'idée de cette approche est d'extraire des entités nommées présentes dans le texte en utilisant l'outil GATE[21], et de les annoter afin d'instancier les concepts d'une ontologie de haut niveau (KIMO) représentée en RDFS. Ces instances sont ensuite utilisées pour annoter les documents et pour enrichir la base de connaissances de KIM. Les annotations créées sont à la fois stockées directement dans le document annoté ainsi que dans un serveur d'annotation RDF.

OntoAnnotate est un outil d'annotation semi-automatique qui permet de rassembler les connaissances des documents et des pages Web aux formats HTML et XML, de créer des annotations sémantiques de document et d'enrichir la base d'annotations avec des méta-données. Cet outil basé sur l'approche CREAM

[19] http://kmi.open.ac.uk/projects/akt/about.html

[20] Traitement Automatique de la Langue Naturelle

[21] Un environnement de développement pour les outils et les applications de TALN - http://nlp.shef.ac.uk/

[Handschuh et al., 2001] utilise la source de données fondamentales et le moteur d'inférence de l'outil OntoBroker[22] pour son processus d'annotation. Par défaut, Annotea utilise les propriétés du DublinCore, telles que l'auteur, la date, le titre, l'éditeur, etc., pour créer des méta-données sur les documents traités. L'utilisateur peut également fournir une ontologie de domaine modélisée en RDFS. Les annotations créées sont conservées de façon externe au document annoté. Ces annotations sont alors disponibles soit localement sur l'ordinateur de l'utilisateur, soit sur un serveur d'annotation RDF public.

OntoMat-Annotizer[23] est un outil d'annotation de page Web qui est intuitif et interactif pour l'utilisateur. Cet outil est aussi basé sur l'approche CREAM. Il aide aux tâches de création et de maintenance des méta-données des pages Web. Au lieu d'annoter manuellement la page avec un éditeur de texte, OntoMat-Annotizer permet à l'annotateur de surligner les parties appropriées de la page Web et de créer de nouvelles instances par les opérations *drag-and-drop*. Cet outil est écrit comme une application Java. Il a une architecture modulaire avec des *plugs-in* qui sont dédiés au navigateur d'ontologie, au navigateur de web, au mécanisme d'inférence, etc.

SHOE Knowledge Annotator[24]**,** développé à l'université de Maryland, est un des premiers outils suivant l'idée d'annoter des pages Web [Heflin, 2001]. Cet outil permet d'annoter sémantiquement les pages HTML à l'aide d'une ontologie modélisée en SHOE [Heflin et Hendler, 2000]. L'utilisateur peut annoter ses documents avec une ou plusieurs ontologies de domaine qui doivent être implémentées en langage SHOE et accessibles soit localement soit sur le Web via une adresse URL. Avec cet outil, les annotations sont conservées directement (i.e. internes) dans le document annoté.

S-CREAM [Handschuh et al., 2002] est une plate-forme pour la création semi-automatique d'annotations sémantiques basées sur une ontologie. Cette plate-forme, basée sur l'approche CREAM, aide à la création des méta-données avec l'aide d'extraction de l'information. Elle fournit certains nouveaux dispositifs tels que l'inférence des services, crawler, le système de gestion des documents, des éditeurs de documents… L'implémentation de S-CREAM est l'outil OntoMat qui a pour but l'annotation semi-automatique basée sur le composant d'extraction d'information Amilcare [Ciravegna, 2003].

[22] http://ontobroker.semanticweb.org/

[23] http://annotation.semanticweb.org/ontomat.html

[24] http://www.cs.umd.edu/projects/plus/SHOE/KnowledgeAnnotator.html

1.1.3 De la Mémoire d'entreprise au Web sémantique d'entreprise

La section précédente nous a apporté une vue d'ensemble sur le principe et les avantages de la vision du Web sémantique. Nous avons aussi examiné l'ontologie et l'annotation sémantique qui sont deux composants importants participant à la réalisation de la vision du Web sémantique.

Nous discutons par la suite l'utilisation des mémoires d'entreprise (ou d'organisation) dans la nouvelle génération de Web sémantique. Comment intégrer les mémoires d'entreprise et les systèmes de gestion des connaissances d'entreprise dans l'environnement de Web sémantique en vue de bénéficier de ses technologies et ses avantages ?

1.1.3.1 Mémoire d'entreprise et Gestion des connaissances

A l'heure actuelle, les sources d'informations et de connaissances dans les entreprises deviennent de plus en plus importantes et elles jouent aussi un rôle crucial pour le développement de l'entreprise. Le partage du travail nécessite un partage de connaissances au sein des entreprises et ce besoin peut devenir encore plus crucial dans l'avenir. Pour réussir, les entreprises doivent bien gérer ces sources de connaissances et supporter l'utilisation effective des connaissances.

Les connaissances développées lors des activités d'une entreprise peuvent être exprimées explicitement dans des actes, des documents, des expériences. Nous appelons *mémoire d'entreprise* ou *mémoire organisationnelle* tous ces types de connaissance, des ressources et aussi les informations cruciales de l'entreprise. [Van Heijst et al., 1996] définit une mémoire d'entreprise comme "la représentation explicite et persistante des connaissances d'une organisation". Ces connaissances peuvent porter, par exemple, sur la stratégie de l'entreprise, les procédés de travail, les clients, etc.

D'après [Pomian, 1996], la construction d'une mémoire d'entreprise repose sur la volonté de "préserver afin de les réutiliser plus tard ou le plus rapidement possible, les raisonnements, les comportements, les connaissances, même en leurs contradictions et dans toute leur variété". En effet, une telle mémoire représente un atout important pour l'entreprise car elle lui permet de pérenniser des connaissances explicites/implicites, de réutiliser de façon meilleure les leçons apprises lors des précédents échecs/succès et de créer de nouvelles connaissances.

Dans le cadre de notre travail, nous utilisons la définition qui est donnée dans [Dieng et al., 2005] : la mémoire d'entreprise comme "la matérialisation explicite et persistante des connaissances et informations cruciales d'une organisation pour faciliter leur accès, partage et réutilisation par les membres de l'organisation dans leurs tâches individuelles et collectives". La gestion d'une

mémoire d'entreprise est considérée comme une partie de la gestion de la connaissance. Selon [Ermine, 2000], le processus de gestion de connaissances d'une organisation s'articule principalement autour de trois points-clés : capitalisation, partage et création de connaissances. [Dieng et al., 2005] définit la gestion des connaissances comme le management des ressources de connaissances d'une organisation pour faciliter :

- l'accès, le partage, la réutilisation de ces connaissances (celles-ci pouvant être explicites ou tacites, individuelles ou collectives), avec un objectif de patrimonial de capitalisation.

- la création de nouvelles connaissances avec un objectif d'innovation.

1.1.3.2 Web sémantique d'entreprise

La construction de la mémoire d'entreprise entraîne de plus en plus l'utilisation effective des connaissances partagées au sein de l'entreprise. Un nombre croissant d'entreprises rendent accessible nombre de documents dans leur intranet, constituant ainsi une mémoire de l'entreprise permettant un partage de connaissances dans l'entreprise. Dans la prochaine génération du Web sémantique, nous nous intéressons aux nouvelles perspectives de la combinaison des technologies du Web sémantique et des moyens de communication de l'entreprise (i.e. l'intranet, l'intraweb) pour caractériser la mémoire d'entreprise. De l'analogie entre les ressources d'une mémoire d'entreprise et celles du Web, l'équipe ACACIA/EDELWEISS a proposé de matérialiser une mémoire d'entreprise sous forme d'un Web Sémantique d'entreprise [Dieng-Kuntz et al., 2005] en vue de mieux découvrir et utiliser les connaissances de cette mémoire.

Cette approche est un des sujets principaux présentés dans ce manuscrit. Par conséquent, nous réservons le chapitre suivant à la description détaillée du Web sémantique d'entreprise et du problème de son évolution.

1.2 Évolution du système de gestion des connaissances

1.2.1 Problème de changements et de l'évolution

Nous avons mentionné dans les sections ci-dessus le fait que de plus en plus d'organisations exploitent un système de gestion des connaissances afin de faciliter l'accès, le partage, et la réutilisation des connaissances organisationnelles. Cependant, l'environnement dynamique, distribué et en cours d'évolution a renforcé les changements dans les organisations ainsi que dans leur système de gestion des connaissances. Ces changements peuvent être classifiés en deux catégories suivantes:

- *Changements des métiers de l'organisation :* Ce type de changements concerne la conceptualisation du domaine métier. Dans les opérations de l'organisation, on peut changer de points de vue sur la conception de certains produits ou sur la manière d'effectuer des opérations. Par exemple, nous développons une ontologie de domaine décrivant les activités d'une organisation. Cette ontologie est utilisée pour modéliser des connaissances organisationnelles dans le système de gestions des connaissances de l'organisation. Elle contient des concepts `Dept_SupportRéseaux` et `Dept_SupportLogiciel` qui modélisent le département du support de réseaux et l'autre du support de logiciel respectivement. Si nous fusionnons ces deux concepts en un nouveau concept `Dept_SupportTechnologie`, cela peut affecter le point de vue sur la hiérarchie des départements du système de gestion des connaissances et cela entraînera des changements continus dans le système.

- *Changements de techniques :* Le deuxième type de changement se passe essentiellement sur les aspects techniques du système. Par exemple, l'ingénieur du système fait quelques modifications sur l'ontologie, sur la base d'annotations ou sur les fonctions d'opération du système (e.g. supprimer un concept de l'ontologie, remplacer une ontologie par une autre nouvelle ontologie, migrer des annotations, etc.). Ces modifications pourraient affecter directement le fonctionnement du système de gestions des connaissances.

Tous ces types de changements doivent être bien gérés au cours de l'évolution du système. Une des exigences importantes pour la gestion des changements et de l'évolution d'un système de gestion des connaissances est d'assurer la consistance et l'exactitude de chaque composant du système ainsi que des relations cohérentes entre les composants après chaque modification dans le système. En plus, les systèmes de gestion des connaissances deviennent de plus en plus grands et complexes en raison de la croissance impressionnante des connaissances ainsi que les fonctions intégrées. Il n'est pas envisageable de laisser à un ingénieur la gestion manuelle du système car il serait très difficile de comprendre tout le système et d'avoir la capacité de contrôler tous les changements apparus dans le système. Par conséquent, on a besoin d'un mécanisme de gestion de l'évolution du système d'une manière automatique ou d'assister l'ingénieur à l'aide de suggestions dans le processus de réalisation des changements (i.e. solution semi-automatique).

Les changements influencent normalement les activités et l'exécution du système de gestion des connaissances entraînant l'évolution du système. Le problème de gestion de l'évolution des changements concernent principalement les trois problèmes suivants:

- *Détecter les changements et les inconsistances :* ce problème demande de chercher tous les changements effectués et ceux qui pourraient entrainer les inconsistances du système.

- *Corriger les inconsistances :* Après avoir détecté les inconsistances, celles-ci doivent être résolues afin de garantir la consistance globale de tout le système.

- *Propager les changements :* Les changements exécutés doivent être propagés aux parties dépendantes pour réparer celles qui sont affectées par ces changements.

Dans la section suivante, nous allons examiner quelques approches existantes essentielles qui ont un lien étroit avec la gestion de l'évolution et des changements du système de gestion des connaissances.

1.2.2 Approches principales sur la gestion de l'évolution

1.2.2.1 Evolution des systèmes à bases de données / bases de connaissances

Les problèmes de l'évolution et de la gestion des versions de schéma de bases de données ont été étudiés intensivement dans le passé. Ces travaux se sont concentrés principalement sur les bases de données orientées objets et sur les bases de données relationnelles. [Roddick, 1995] distingue :

- *Évolution de schéma :* un système de bases de données soutient l'évolution de schéma s'il facilite la modification du schéma de bases de données sans perte de données existantes.

- *Gestion des versions de schéma (versioning):* un système de bases de données soutient la gestion des versions de schéma s'il facilite les requêtes de données à travers plusieurs versions différentes de schéma.

Un état de l'art assez complet des recherches existantes sur l'évolution et la gestion des versions de schéma est aussi présenté dans [Roddick, 1995]. Les auteurs de [Franconi et al., 2000] arguent que l'évolution de schéma peut être considérée comme un cas spécial de la gestion des versions de schéma dans le contexte où seule la version courante de schéma est maintenue. Ils ont présenté également les problèmes de (i) sémantique du changement et (ii) propagation du changement. Le premier problème concerne le traitement des changements de schéma et leur effet sur le schéma de bases de données afin de maintenir la consistance de schéma. Le deuxième problème a pour but de propager les changements aux données réelles pour aussi préserver la consistance du schéma modifié.

Afin de réaliser la sémantique du changement, les recherches existantes suivent essentiellement deux approches. La première approche repose sur des invariants et des règles. Les invariants définissent les conditions qui doivent être satisfaites pour assurer la consistance du schéma, les règles sont utilisées pour reconstituer la consistance lorsque des invariants sont violés après un changement. La deuxième approche est basée sur l'introduction d'axiomes formalisant l'évolution dynamique de schéma. Ces axiomes (avec un mécanisme d'inférence) assurent qu'un schéma évoluera vers une version cohérente, sans besoin de vérifier les inconsistances.

Une autre étude [Ahmed-Nacer, 1994] apporte une solution aux aspects de gestion et d'évolution des schémas de bases d'objets logiciels. Les auteurs proposent un modèle fondé sur la généricité de la politique d'évolution des schémas pour permettre d'adapter, à chaque type d'application, la stratégie d'évolution souhaitée. Ce modèle utilise un mécanisme de gestion des points de vue et de cohérence de la base commune d'objets, et considère les schémas comme des configurations de types.

[Bounaas, 1995] présente une solution aux problèmes d'extension et de réutilisation des mécanismes d'évolution de schéma dans les bases de connaissances. Pour l'évolution de schémas, les auteurs ont opté pour une approche par correction où les classes sont immédiatement modifiées (sans création de versions) et où toutes les instances sont converties immédiatement après la modification des classes. Ils proposent un système d'évolution SHOOD permettant la définition et la mise en œuvre de la dynamique. Cette mise en œuvre est réalisée par un ensemble de mécanismes tels que la classification d'instances et les règles actives ou règles ECA (Evénement, Condition, Action), ainsi qu'un ensemble d'opérations de manipulation: le support d'évolution. Ils proposent de même un mécanisme de règles et de stratégies d'évolution pour permettre une expression déclarative des contraintes d'évolution de structures ou de données.

1.2.2.2 Évolution de l'ontologie

Les ontologies doivent normalement changer pour pouvoir s'adapter aux conditions métiers changées. Les disparités (mismatches) entre les ontologies sont le problème principal qui gêne l'utilisation combinée des ontologies indépendamment développées. Certaines recherches distinguent plusieurs types de disparités qui peuvent se produire entre différentes ontologies [Klein, 2004].

Nous avons présenté dans la section précédente certaines approches de recherche sur l'évolution de schéma dans les bases de données. En se basant sur ces recherches, [Stojanovic, 2004] distingue les terminologies sur la gestion de l'ontologie, la modification, l'évolution et la gestion des versions:

- *Gestion de l'ontologie :* Il s'agit d'un ensemble des méthodes et techniques qui sont élaborées pour utiliser efficacement de multiples

variantes des ontologies, éventuellement à partir de sources différentes en vue de résoudre des tâches différentes. Par conséquent, un système de gestion d'ontologie est considéré comme un cadre pour créer, modifier, gérer des versions, faire des requêtes et sauvegarder des ontologies, etc.

- *Modification de l'ontologie :* le système de gestion de l'ontologie permet d'effectuer des changements sur l'ontologie considérée, sans compter son état de consistance.

- *Évolution de l'ontologie :* le système de gestion de l'ontologie facilite la modification d'une ontologie en préservant son état de consistance.

- *Gestion des versions de l'ontologie (versioning):* le système de gestion de l'ontologie permet la manipulation des changements de l'ontologie en créant et en gérant ses différentes versions.

Dans la dernière décennie, la majorité des recherches sur le domaine de l'ingénierie de l'ontologie s'est focalisé principalement sur le problème de construction de l'ontologie. Il y a peu de recherches qui étudient sur le problème du traitement des changements et de la gestion de l'évolution de l'ontologie. Parmi les études sur les changements de l'ontologie ci-dessus, les deux approches (i) évolution de l'ontologie et (ii) la gestion des versions de l'ontologie ont attiré le plus de travaux actuels.

Dans [Heflin, 2001], les auteurs précisent que les ontologies sur le Web doivent évoluer. Ils fournissent une nouvelle définition formelle des ontologies pour l'usage dans les environnements dynamiques et distribués, ils présentent également SHOE, un langage de représentation des connaissances sur le Web.

[Pinto et al., 2004] propose un modèle pour gérer la négociation des changements provenant des acteurs distants qui tentent de modifier une ontologie partagée. Ce modèle aborde l'évolution de l'ontologie dans un environnement multi-acteurs dans lequel chaque utilisateur modifie l'ontologie à partir de son poste individuel, en construisant ainsi sa propre ontologie locale. Un comité scientifique analyse ensuite les ontologies locales pour identifier les changements à introduire dans l'ontologie partagée.

[Stojanovic et al., 2002] définit des notions nécessaires pour l'évolution de l'ontologie, la consistance de l'ontologie et la taxonomie des changements de l'ontologie. Les auteurs abordent également les conditions pour un système d'évolution de l'ontologie et dérivent un processus d'évolution de six phases qui les satisfait [Stojanovic, 2004]. Ce processus analyse systématiquement les causes et les conséquences des changements, et assure la consistance de l'ontologie ainsi que ses parties dépendantes après la résolution des changements ontologiques. Stojanovic et al. présentent une approche procédurale et une approche déclarative pour résoudre des changements de l'ontologie, des solutions pour propager des changements dans l'environnement local et distribué. Ils introduisent aussi le

langage d'ontologie et le prototype KAON proposé par l'Université de Karlsruhe pour un système d'évolution de l'ontologie. Cette approche de l'évolution de l'ontologie a bien traité le problème de résolution des changements de l'ontologie vers les parties ontologiques dépendantes. Cependant, d'après [Rogozan, 2004], l'approche proposée par [Stojanovic, 2004] a également montré certaines limites. Premièrement, les auteurs ne proposent aucune étape d'analyse des effets des changements sur la relation de compatibilité entre l'ontologie évoluée et les artefacts dépendants. En ce sens, l'étape de propagation des changements, proposée par les auteurs, est assez unidirectionnelle vu qu'elle vise seulement la modification des ontologies dépendantes afin de préserver leur consistance avec l'ontologie évoluée. Deuxièmement, les auteurs développent un processus d'évolution qui ne prend pas en compte la gestion des versions multiples.

Les approches plus récentes [Plessers, 2006] et [Flouris, 2006] se focalisent également sur l'aspect de l'évolution de l'ontologie. Cependant, ces deux approches s'intéressent au modèle logique de l'ontologie en proposant des solutions pour maintenir la consistance logique. [Plessers, 2006] propose de construire un langage de définition des changements (Change Definition Language) pour représenter formellement des changements de l'ontologie. Il utilise aussi le journal d'évolution afin de conserver toutes les versions des concepts au cours de l'évolution. Dans [Flouris, 2006], les auteurs proposent la notion de changement de croyance (belief change) et la théorie AGM pour résoudre le problème de conformité de représentation des connaissances, et cela est appliqué en particulier au cas des ontologies en OWL.

1.2.2.3 Gestion des versions de l'ontologie

Influencés par les recherches relatives à l'évolution et la gestion des versions de schéma dans les bases de données à objets-orientées et relationnelles, certains travaux actuels considèrent la gestion des versions de l'ontologie comme une variante de l'évolution de l'ontologie [Haase et Sure, 2004]. Sous ce point de vue, l'évolution de l'ontologie est concernée par la capacité de modifier l'ontologie sans perte de données et permettant d'accéder aux données via la dernière version de l'ontologie, tandis que la gestion des versions de l'ontologie permet de créer les différentes versions et d'accéder aux données à travers ces versions. Dans un autre travail [Flouris, 2006], les auteurs différencient la gestion des versions et l'évolution de l'ontologie. Ils considèrent l'évolution comme le processus de modification de l'ontologie en maintenant sa validité, tandis que la gestion des versions est le processus par lequel sont gérées plusieurs versions de l'ontologie en maintenant l'interopérabilité entre ces versions et permettant d'accéder à chaque version selon les exigences de l'élément d'accès (données, service, application ou une autre ontologie). [Heflin, 2001] aborde le support de multiples versions d'ontologies en permettant d'associer à chaque version ontologique un identifiant unique ainsi qu'une balise *"Backward-Compatible_With"* spécifiant les versions compatibles ou rétrocompatibles. Cette

approche supporte aussi la réutilisation d'ontologies mais il ne traite pas de la propagation des changements aux ontologies distribuées et dépendantes.

Un problème important de la gestion des versions de l'ontologie est la "relation de versions" entre les éléments ontologiques (i.e. les classes, les propriétés) apparaissant dans différentes versions de l'ontologie. Cette relation est appelée la spécification de changement dans [Klein et Fensel, 2001] et son rôle est de rendre le lien entre différentes versions des éléments ontologiques explicites. Grâce à cette spécification, nous pouvons identifier les changements effectués et les éléments affectés de l'ontologie. Il y a plusieurs manières de représenter cette spécification de changement. [Zhang et al., 2003] définit une "spécification de migration" qui associe des concepts entre différentes versions d'une ontologie après chaque exécution d'un changement. [Noy et Musen, 2004] utilise un "structural diff" pour garder les traces de changements entre deux versions de l'ontologie. [Plessers et Troyer, 2005] propose d'utiliser un "journal de version" pour sauvegarder les différentes versions de chaque élément ainsi que la relation entre elles et quelques méta-données relatives représentant l'exécution des changements.

En réalité, quelques applications peuvent continuer à utiliser les anciennes versions de l'ontologie et à mettre à jour leur propre version (ou pas du tout). Sous ce point de vue, d'après [Noy et Klein, 2004], l'évolution et la gestion des versions de l'ontologie deviennent indistinguables en raison de l'environnement distribué et décentralisée du Web sémantique. Les multiples versions des ontologies liées doivent être supportées. [Klein, 2004] définit un framework pour la gestion des versions de l'ontologie en décrivant ses éléments principaux : une méta-ontologie des opérations de changement, des opérations de changement complexes, un ensemble de transformations, un *template* pour la spécification de changement. Les auteurs présentent aussi le vocabulaire et décrivent un certain nombre d'hypothèses de base qui sont utilisées dans leur approche, alors ils expliquent les éléments de base du framework et les relations entre eux, et finalement indiquent comment ces éléments peuvent être opérationnalisés. [Klein, 2004] fait également une distinction entre les différents niveaux d'interprétation d'une ontologie, elle est utile pour distinguer différents types de changements d'ontologie. Un inconvénient de cette approche, d'après [Rogozan, 2004], est qu'ils fournissent un modèle d'analyse de la relation entre les versions de l'ontologie, mais sans se préoccuper de la gestion de l'accès aux artefacts dépendants (i.e. objets référencés, ontologies, applications) au moyen de versions de l'ontologie. De plus, les auteurs ne développent aucun cadre fonctionnel pour intégrer la totalité des éléments méthodologiques qu'ils proposent.

D'autre part, les auteurs de [Huang et Stuckenschmidt, 2005] proposent une approche de raisonnement sur les différentes versions de la même ontologie afin de vérifier la compatibilité pour les applications existantes. Bien que leur approche permette de raisonner sur différentes versions, ils ne laissent pas spécifier les conditions de compatibilité qui peuvent être utilisées pour vérifier automatiquement la compatibilité d'une version de l'ontologie avec une autre

version précédente. Cette approche est similaire à celle proposée par [Plessers, 2006], elles se basent sur une logique temporelle.

1.2.2.4 Évolution des méta-données et des annotations

Les méta-données constituent une voie pour aider l'utilisateur ou le gestionnaire d'information à comprendre, retrouver, comparer des informations sans forcément avoir recours directement au contenu de celles-ci. Malgré la différence de structure, tous les types de méta-données poursuivent un objectif commun : offrir des éléments de description pour faciliter l'accès à des ressources données en fournissant toute l'information les concernant [Prié et Garlatti, 2004]. Cependant, les ressources données peuvent être changées à cause du changement des conditions métiers ou des objectifs d'utilisation. Cela pourrait affecter les méta-données qui décrivent ces ressources. Dans le contexte du Web sémantique, un autre facteur pourrait entraîner l'évolution des méta-données : c'est le changement de l'ontologie. Jusqu'à présent, il y a peu de recherches sur l'évolution des méta-données en générale et sur des annotations sémantiques en particulier.

[Stojanovic et al., 2002b] et [Handschuh, 2005] présentent une approche permettant d'assurer la consistance de la description des sources de connaissances dans un système de gestion de connaissance basé sur l'ontologie. Les auteurs se concentrent sur l'évolution des méta-données en présentant l'architecture du système CREAM. Le processus du système de l'évolution des méta-données CREAM peut être représenté dans les phases principales suivantes :

- *Capture des méta-données :* Quand une ontologie est modifiée, les instances doivent être changées de telle manière que l'ontologie et les instances restent conformes l'une à l'autre. Si les instances sont sur le Web, elles sont rassemblées en base de connaissance à l'aide d'outils comme le *crawler*. Afin d'accélérer tout processus de propagation des changements, seules les instances qui dépendent du changement sont recueillies.

- *Analyse des méta-données :* Dans la deuxième étape, une traduction automatique des instances est exécutée selon les changements de l'ontologie. Cette étape fournit un résultat sous forme de liste d'instances modifiées en référence à la ressource correspondante (source de connaissances).

- *Génération d'une proposition pour des modifications :* dans la dernière étape, les instances "obsolètes" sur le Web sont remplacées par les instances "mises à jour" correspondantes. Certaines modifications des instances peuvent être mises à jour automatiquement, mais pour d'autres instances qui sont "protégées en écriture" il faut avoir la notification

envoyée à l'auteur de l'annotation afin de l'informer au sujet des changements et suggérer comment corriger l'instance.

Une autre approche pour préserver la sémantique de méta-données dans le cas de la maintenance et du raffinement du modèle ontologique est présentée dans [Ceravolo et al., 2004]. Les auteurs décrivent ce problème en se référant à un scénario où une base de données relationnelle est utilisée pour sauvegarder et mettre à jour des ontologies et les méta-données dans le format RDF(S).

Une recherche mentionne également la mise à jour des annotations décrivant des documents pédagogiques d'un système d'apprentissage sur le Web sémantique [Rogozan et Paquette, 2005]. Dans leur contexte de recherche, l'ontologie est utilisée comme référentiel sémantique pour les éléments pédagogiques.

Nous allons présenter les aspects les plus importants de ces travaux ci-dessus d'une manière plus détaillée dans le chapitre suivante.

1.2.2.5 Outils de support pour l'évolution et la gestion des versions de l'ontologie

Nous nous intéressons dans cette partie à certains outils de support qui sont les plus connus et les plus utilisés actuellement pour l'évolution et la gestion des versions de l'ontologie.

PROMPT[25] [Noy et Musen, 2002] [Noy et al., 2004] est une suite de *plugs-in* pour l'outil Protégé en vue de gérer de multiple ontologies. Cet outil a les fonctionnalités principales suivantes : (i) comparer deux versions de l'ontologie, (ii) fusionner deux ontologies et (iii) extraire une partie de l'ontologie. L'outil de gestion des versions (et algorithme) important intégré dans cet outil est PROMPTDiff qui utilise des heuristiques pour comparer différentes versions de l'ontologie au niveau structurel et créer une "structural diff" des différences entre ces versions. Les auteurs de [Noy et Musen, 2003] ont aussi proposé une suite de PROMPT (i.e. PROMPT suite) contenant des outils tels que AnchorPROMPT, iPROMPT, PROMPTDiff, PROMPTFactor afin de réaliser des tâches différentes, i.e. fusion de l'ontologie, alignement de l'ontologie, gestion des versions de l'ontologie, etc.

KAON[26] [Stojanovic et al., 2002a] [Maedche et Staab, 2003] [Gabel et al., 2004] fournit plusieurs fonctionnalités importantes pour la création, la récupération, la maintenance et l'évolution de l'ontologie. Les modules principaux de KAON sont groupés en deux parties (i) le "KAON Front-End" qui définit les interfaces pour communiquer avec l'utilisateur et (ii) "KAON Core" qui implémente des fonctions principales du système. Un outil important de Front-

[25] http://protege.stanford.edu/plugins/prompt/prompt.html

[26] Karlshrue Ontology and Semantic Web framework - http://kaon.semanticweb.org/

End est le “KAON Portal” fournissant certains dispositifs pour la génération des portails du Web basé sur l’ontologie [Gabel et al., 2004]. “KAON Core” fournit les librairies APIs (i.e. KAON API and RDF API) définissant un ensemble d’interfaces qui permettent à des applications externes d’accéder aux fonctionnalités du système. D’autre part, elles servent aussi à implémenter des processus d’évolution et les stratégies d’évolution de l’ontologie [Maedche et al., 2003] [Stojanovic, 2004].

OntoView[27] (ou RDFDiff) [Klein et al., 2002] est un système basé sur le Web qui fournit les fonctionnalités basiques pour gérer les versions de l’ontologie. Ce système est inspiré du système CVS[28] et a ensuite évolué pour supporter le *toolkit* de Jena [McBride, 2001] et l’entrepôt en Sesame RDF [Broekstra et al., 2002]. Il est dédié à la comparaison des versions de l’ontologie représentée en RDF(S). OntoView compare les versions d’une ontologie au niveau structurel, c’est-à-dire au niveau de définitions de concepts et propriétés, afin de découvrir les définitions modifiées d’une version à l’autre [Klein et al., 2002a]. Pour chaque définition modifiée, OntoView produit ensuite une liste de changements considérés comme étant nécessaires pour passer de la définition de l’entité ontologique en V_N à la définition de cette entité en V_{N+1}.

1.3 Conclusion

Nous venons de présenter dans ce chapitre la notion du ***Web sémantique,*** nouvelle génération du Web actuel. La Figure 7 dresse une vue globale sur l’évolution du Web actuel, essentiellement *syntaxique* avec un contenu qui reste quasi *inaccessible aux traitements automatiques* par machines, vers la nouvelle infrastructure du Web sémantique qui a pour ambition d’enlever ces difficultés en ajoutant de la *sémantique* aux données pour permettre de les *traiter automatiquement* par machine. Nous avons présenté des principes (i.e. l’architecture structurée en couches), des technologies (i.e. les modèles et langages de représentation) ainsi que des composants principaux (i.e. ontologie, annotation sémantique, ressource) du Web sémantique et nous avons souligné également plusieurs recherches et applications développées sur ce dernier.

Du point de vue des organisations, quel intérêt apporte le Web sémantique? Cette question rejoint les travaux réalisés dans le domaine de la gestion des connaissances et en particulier dans le domaine des mémoires d’entreprise ou d’organisation. Ces travaux intègrent les techniques du Web sémantique (ontologies, annotations sémantiques et langages formels de représentation, etc.) pour construire le ***Web sémantique d’entreprise*** visant à faciliter son accès, son partage et sa réutilisation des connaissances organisationnelles. Cependant, les organisations ont besoin d’évoluer afin de s’adapter aux changements générés

[27] http://ontoview.org/

[28] Concurrent Versioning System

dans l'environnement du Web sémantique dynamique, distribué et en cours d'évolution. Nous avons fait une synthèse de quelques approches concernant l'évolution des systèmes de gestion des connaissances. Nous aperçevons que les deux composants importants, i.e. les ontologies et les annotations sémantiques, dans ces systèmes, auraient besoin d'être changés pour s'adapter à l'évolution de l'organisation concernée.

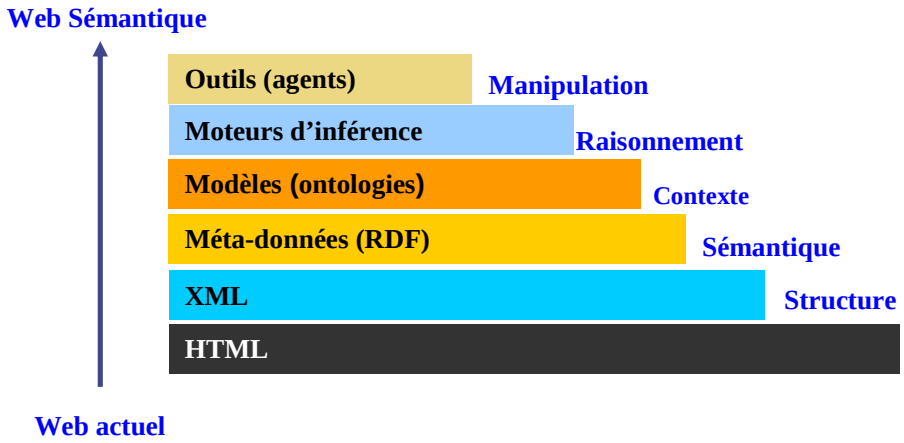


Figure 7 - Evolution du Web actuel vers le Web sémantique

Par conséquent, nous réservons le chapitre suivant à l'approfondissement de certains aspects de l'évolution, particulièrement sur l'ontologie et l'annotation sémantique. Nous positionnons notre travail par rapport aux recherches existantes sur les mêmes aspects.

Chapitre 2

Évolution du Web Sémantique d'Entreprise

La gestion des connaissances dans une entreprise a pour objectif de favoriser la croissance, la transmission et la conservation des connaissances dans cette entreprise. Dans la prochaine génération du Web, le Web sémantique visant une meilleure coopération entre humains et machines [Berners-Lee et al., 2001], l'approche de la gestion des connaissances de l'entreprise consistera en la construction d'un Web sémantique d'entreprise (WSE) afin de faciliter l'accès, le partage, et la réutilisation des connaissances organisationnelles [Dieng-Kuntz, 2005]. Cependant, les organisations vivent dans un environnement dynamique et en cours d'évolution qui mène souvent à des changements de leur système de gestion des connaissances. Dans ce chapitre, nous présentons brièvement l'approche du WSE et les causes du problème de l'évolution du WSE. Nous nous intéressons aux scénarios d'évolution de l'ontologie et de l'annotation sémantique qui sont deux composants importants dans un système de gestion des connaissances basé sur le Web sémantique. Dans ce sens, nous proposons un aperçu des aspects principaux de l'évolution de l'ontologie et de l'annotation sémantique dans le cadre de notre présent travail.

2.1 Introduction

Dans le chapitre précédent, nous avons présenté la construction d'une mémoire organisationnelle ou mémoire d'entreprise comme une approche pour la gestion des connaissances permettant à une organisation (ou une entreprise) de faciliter la préservation et l'utilisation de ses connaissances organisationnelles. Comment devrait-elle changer la mémoire d'entreprise - en particulier, si celle-ci repose sur le Web sémantique - pour mieux s'adapter aux nouveaux besoins et aux exigences de l'entreprise?

Nous examinons dans cette section la matérialisation d'une mémoire d'entreprise vers un Web sémantique d'entreprise (WSE) dans un environnement hétérogène et distribué qui peut entraîner des incohérences dans le système. Ces inconsistances ont besoin d'être détectées et corrigées. Pour ce faire, nous allons étudier les facteurs causant le problème d'évolution du WSE en examinant certains aspects importants au cours de l'évolution et particulièrement sur les deux composants principaux du WSE, à savoir l'ontologie et l'annotation sémantique.

2.1.1 Web sémantique d'entreprise - Approche Acacia

Les auteurs dans [Dieng-Kuntz et al., 2005] abordent une approche particulière de la gestion des connaissances qui est la constitution d'une mémoire d'entreprise ou mémoire organisationnelle, matérialisant et indexant les connaissances et informations cruciales de l'organisation (i.e. une entreprise, une institution ou une communauté de pratique) afin d'améliorer leur accès, partage et réutilisation voire de permettre la création de nouvelles connaissances par les membres de l'organisation dans leurs tâches individuelles et collectives. La construction d'une telle mémoire d'entreprise entraîne de plus en plus l'utilisation effective des connaissances partagées au sein de l'entreprise. Avec l'évolution du Web vers le Web sémantique, les nouvelles technologies pour ce dernier peuvent aussi être appliquées pour l'organisation. Les ressources de cette mémoire d'entreprise sont similaires à celles du Web, elles ont besoin d'être annotées sémantiquement pour mieux découvrir et utiliser les connaissances de ces ressources. Partant de l'analogie entre les ressources d'une mémoire d'entreprise et celles du Web, l'équipe ACACIA[29] a proposé de matérialiser une mémoire d'entreprise sous forme d'un Web Sémantique d'entreprise [Dieng-Kuntz, 2005]. Les principales composantes de ce web sémantique d'entreprise sont les suivantes :

- *Les ressources :* il peut s'agir de bases de données, de documents (dans toutes sortes de formats), de services/logiciels, voire de personnes, etc.

[29] http ://www-sop.inria.fr/acacia/

- *Les ontologies :* elles décrivent le vocabulaire partagé par les différentes communautés de l'entreprise.

- *Les annotations sémantiques :* elles décrivent des méta-données sur les ressources en se basant sur les concepts et les relations de l'ontologie.

La Figure 8 présente l'architecture d'un Web sémantique d'entreprise dans lequel les technologies du Web sémantique sont utilisées pour réaliser les tâches du Web sémantique d'entreprise. Dans cette architecture, nous avons des ontologies communes décrivant d'une manière formelle des connaissances du domaine de l'entreprise. Les annotateurs annotent sémantiquement des ressources d'entreprise telles que la description du contenu des documents, les compétences des personnes, les caractéristiques des services… en utilisant le vocabulaire commun et partagé au sein de l'entreprise par des ontologies. Avec l'aide d'un système de gestion des connaissances, les utilisateurs (individuels et collectifs) dans l'entreprise peuvent alors utiliser des annotations créées grâce aux significations des termes, des concepts et des propriétés prédéfinies dans l'ontologie. Cela permet aux anciens utilisateurs de partager leurs connaissances, leurs expériences et aux nouveaux utilisateurs d'apprendre, de consulter et de mieux s'adapter à l'entreprise.

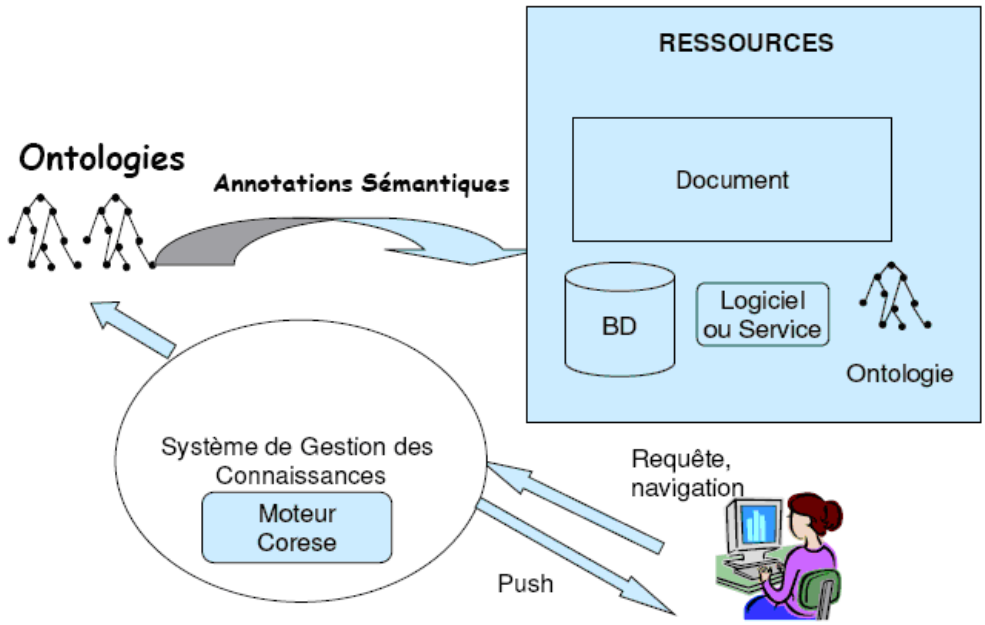


Figure 8 - Architecture d'un Web sémantique d'entreprise [Dieng et al., 2004]

2.1.2 Cycle de développement

Le cycle de développement de la mémoire d'entreprise se décompose en six étapes (c.f. Figure 9) principales suivantes [Dieng-Kuntz et al., 2005] :

- *Besoins (détection des besoins) :* détermination des utilisateurs et de leurs besoins, d'un modèle d'exploitation utile et adapté à l'environnement de travail des utilisateurs. Cette phase définit aussi des scénarios d'utilisation de la mémoire.

- *Conception (construction de la mémoire d'entreprise) :* identification et choix des sources de la mémoire (documents, bases de données…), détermination des connaissances à prendre en compte, choix et techniques pour la matérialisation de la mémoire (mémoire documentaire, mémoire à base de connaissances…).

- *Diffusion :* détermination d'un scénario et du mode de diffusion, utilisation de l'Intranet de l'entreprise par exemple pour diffuser les connaissances via des collecticiels ou des serveurs de connaissances.

- *Utilisation :* utilisation des fonctionnalités de la mémoire d'entreprise, par exemple pour rechercher des informations stockées dans la mémoire en utilisant des moteurs de recherche ou pour disséminer proactivement des informations vers les utilisateurs selon leurs centres d'intérêt.

- *Évaluation :* évaluation de la mémoire d'entreprise selon les critères des utilisateurs, organisationnels ou technologiques.

- *Évolution (maintenance) :* évolution de la mémoire d'entreprise suite à des modifications de composants ou à l'interaction entre des composants.

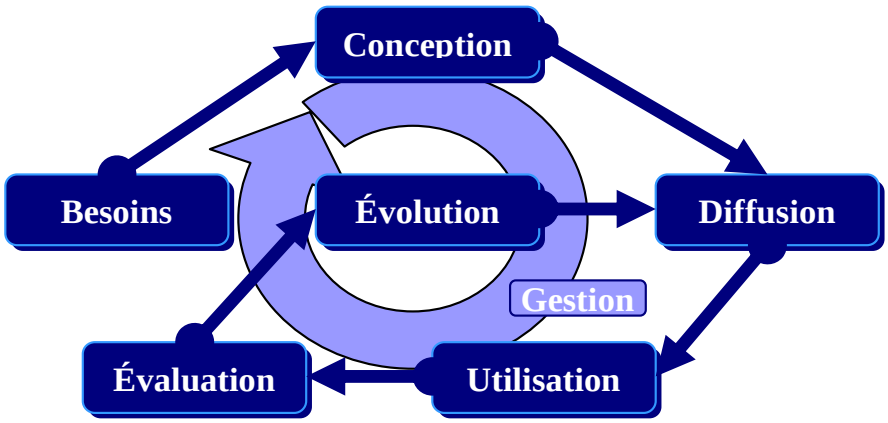


Figure 9 - Le cycle de vie de la mémoire d'entreprise [Dieng et al., 2001]

Dans ce manuscrit de thèse, nous nous intéressons à la phase d'Évolution et de maintenance qui joue un rôle important pour permettre d'adapter la mémoire d'entreprise aux nouveaux besoins et contextes ainsi qu'aux changements des composants internes au cours de l'évolution afin de la garder à jour par rapport à la stratégie de l'entreprise. Le processus d'évolution assure que les modifications de la mémoire seront réalisées d'une manière dynamique et consistante.

2.1.3 Applications du Web Sémantique d'entreprise

Nous présentons dans cette section des projets basés sur l'approche du Web sémantique d'entreprise et réalisés au sein de l'équipe ACACIA.

Samovar

Samovar [Golebiowska et al., 2001] est un système/méthode de capitalisation de connaissances dans le domaine automobile. L'objectif de ce projet était d'améliorer l'exploitation des informations stockées dans un système de gestion de problèmes afin de les mettre à disposition pour les projets futurs chez Renault.

L'approche Samovar repose sur l'utilisation de plusieurs ontologies (Problème, Pièce, Prestation et Projet) construites semi-automatiquement à partir de sources différentes, à savoir (a) les interviews des experts de Renault, (b) les données structurées contenues dans les bases de données et (c) les données textuelles constituées par les discussions des concepteurs stockées dans des champs textuels de la base de données. Ces ontologies sont représentées en RDFS et servent à créer des annotations RDF sur la base des descriptions des problèmes, et à faciliter la recherche d'informations en utilisant le moteur de recherche sémantique Corese.

CoMMA

CoMMA (Corporate Memory Management through Agents) est un projet européen qui a permis de construire une mémoire d'entreprise matérialisée dans une base documentaire annotée par des annotations sémantiques basées sur l'ontologie O'CoMMA [Gandon, 2002]. Ce projet visait à développer une société d'agents coopérant et guidés par l'ontologie pour la recherche d'information dans cette mémoire d'entreprise, l'ajout d'annotations dans la base d'annotations et l'interaction avec les utilisateurs en intégrant un moteur de recherche sémantique.

Le projet CoMMA a traité aussi l'aspect distribué de la mémoire d'entreprise. Des agents dédiés aux connexions ont été proposés, ces agents proposaient des services de pages jaunes et de pages blanches pour fournir des informations sur un agent pouvant offrir un service particulier.

CORESE

CORESE (COnceptual REsource Search Engine) est un moteur de recherche sémantique dédié au langage RDF(S) et qui peut être utilisé pour interroger les différentes ressources d'un web sémantique d'entreprise [Corby et al., 2004]. Ce moteur de recherche rend possible les inférences sur des annotations en RDF basées sur une ontologie. Il repose sur une correspondance entre RDF(S) et le formalisme des graphes cnonceptuels. Il dispose en outre d'une base de règles [Corby et al, 2006] et permet des requetes SPARQL [Corby et Faron, 2007].

MEAT

MEAT (Mémoire d'Expériences pour l'Analyse du Transcriptome) est un projet de construction d'une mémoire d'expériences, qui vise à faciliter la validation et l'interprétation des expériences puces à ADN. L'objectif principal de cette mémoire est d'organiser les connaissances de ce domaine, qui proviennent de plusieurs sources hétérogènes (documents, base de données, connaissances humaines…) afin de faciliter leur partage et leur réutilisation. Ce projet vise donc à offrir aux biologistes un accès transparent et 'intelligent' à l'ensemble de ces connaissances [Khelif, 2006].

OntoWatch

Le projet OntoWatch s'est déroulé dans le cadre d'une coopération entre le Centre Scientifique et Technique du Bâtiment (CSTB) et l'équipe ACACIA de l'INRIA Sophia Antipolis. L'objectif principal était d'exploiter les technologies du Web Sémantique pour développer un système de veille (OntoWatch), guidé par des ontologies, pour collecter, capturer, filtrer, classer et structurer le contenu du Web en provenance de plusieurs sources d'information dans un scénario d'aide à la veille technologique et scientifique [Cao, 2006].

2.2 Problèmes d'évolution du WSE

La mémoire organisationnelle est un système distribué au sens large du terme, elle a cela en commun avec le Web qu'elle est un paysage d'informations hétérogènes et distribuées [Gandon, 2002]. Le système de gestion des connaissances de l'organisation reposant sur une telle mémoire a aussi les mêmes caractéristiques. Le fait d'avoir un environnement hétérogène et distribué avec les changements internes ou externes possibles, implique, la plupart du temps, des changements continus dans le système. Ces changements constituent l'évolution du système de gestion des connaissances.

Un système de gestion de connaissances reposant sur un Web sémantique d'entreprise s'appuie sur le domaine d'activité de l'entreprise. Si le domaine, les fonctions ou l'ordre d'exécution du processus des services de l'entreprise changent, le système de gestion des connaissances devrait aussi évoluer pour s'adapter aux nouveaux besoins ou exigences. Par exemple, les ontologies doivent normalement changer pour s'adapter aux conditions métiers changées. Plusieurs problèmes apparaissent quand on essaie d'utiliser ensemble des ontologies développées indépendamment, ou quand des ontologies existantes sont adaptées pour des nouveaux objectifs. Cependant, non seulement les changements du domaine ou ceux de nouvelles exigences de l'entreprise (c.f. Figure 10), mais aussi les changements internes et les interactions entre des composants pourraient aussi influencer le fonctionnement du système. Dans l'architecture du web sémantique d'entreprise (c.f. Figure 8), les trois composants principaux (i) les ontologies, (ii) les annotations et (iii) les ressources, pourraient changer, entraînant éventuellement l'inconsistance entière ou partielle du système.

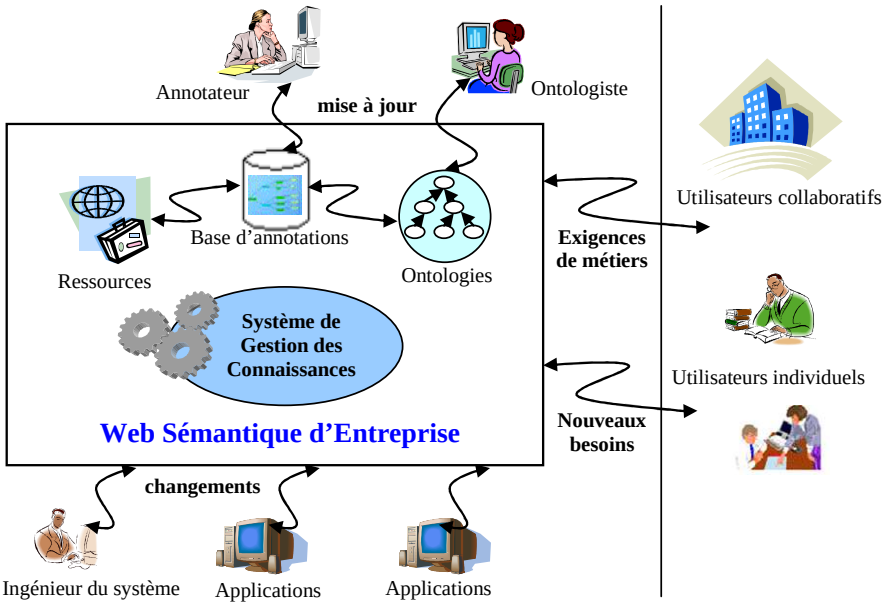


Figure 10 - Contexte de changements

Dans cette section, nous allons examiner les problèmes à l'origine de l'évolution du Web sémantique d'entreprise et particulièrement les causes des changements de chacun des composants du système ainsi que les changements générés par la modification interactive entre ces composants.

Nous présentons également des scénarios d'évolution qui peuvent affecter la consistance du Web sémantique d'entreprise. Nous nous intéressons aux scénarios d'évolution de l'ontologie et de l'annotation sémantique en lien étroit avec notre recherche sur la gestion de l'évolution de l'ontologie et des annotations sémantiques inconsistantes.

2.2.1 Causes possibles des problèmes d'évolution

2.2.1.1 Changements sur chaque composant

Nous examinons les changements possibles des trois composants suivants : l'ontologie, l'annotation sémantique et la ressource (c.f. Figure 11).

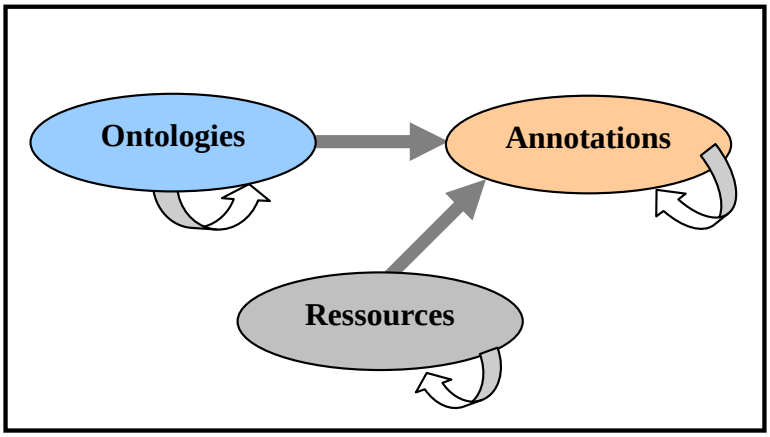


Figure 11 - Changements des composants du WSE

Changement de l'ontologie

Les changements de l'ontologie sont importants pour une application basée sur l'ontologie parce qu'ils affectent la manière dont les données devraient être interprétées et manipulées. Dans la perspective des utilisateurs, ces derniers comptent recevoir des services continus et de haute qualité des applications basées sur les ontologies. Donc, les conséquences et les effets des changements ontologiques doivent être bien gérés.

On peut classer de plusieurs manières les causes des changements de l'ontologie. La première classification est basée sur la nature d'une définition de l'ontologie. D'après [Gruber, 1993], l'ontologie est la spécification de la conceptualisation d'un domaine. Les causes des types de changements de l'ontologie sont donc affectées par les différents niveaux d'interprétation d'une ontologie [Klein and Fensel, 2001] :

- *Changement conceptuel :* un changement dans la conceptualisation ;
- *Changement de spécification :* un changement des spécifications d'une conceptualisation ;
- *Changement de représentation :* un changement de la représentation des spécifications d'une conceptualisation.

Concernant ces différents niveaux d'interprétation des changements de l'ontologie, un autre travail [Flouris, 2006] a traité aussi deux types de changements sur la conceptualisation et sur le domaine dans son approche d'évolution de l'ontologie. Ces deux types de changements ne sont pas rares. La conceptualisation du domaine peut changer pour plusieurs raisons : une nouvelle observation, un changement de point de vue ou une utilisation différente de l'ontologie, un nouvel accès à des informations qui étaient précédemment inconnues, etc. Le domaine lui-même peut également changer parce que le monde réel lui-même n'est généralement pas statique mais évolue à travers le temps.

Les causes des types de changements de l'ontologie sont également distingués par les niveaux de disparités dans une ontologie [Klein, 2004] : niveau du langage et niveau d'ontologie (niveau du modèle). Les disparités entre les ontologies constituent le principal type de problème qui gêne l'utilisation combinée des ontologies indépendamment développées.

- *Disparité au niveau du langage* qui se compose de primitives pour spécifier une ontologie. Les disparités à ce niveau sont celles du mécanisme pour définir des classes, des relations.

- *Disparité au niveau de l'ontologie (ou disparité de modèle)* concerne le domaine décrit dans l'ontologie. Une disparité à ce niveau représente les différences possibles dans la manière de modéliser le domaine.

Les auteurs dans [Visser et al., 1997] distinguent aussi ces deux niveaux mais sous différents noms, respectivement non-sémantique et sémantique. Quant à la fréquence de ces deux types de disparités dans l'ontologie, d'après [Klein, 2004], les disparités au niveau du langage se produisent probablement moins fréquemment que celles du niveau de l'ontologie car les ontologies évolutives sont habituellement exprimées dans un langage.

D'autre part, une autre raison à l'origine des changements de l'ontologie est la modularisation. Les ontologies sont construites normalement d'une manière distribuée et modularisée. Du fait de la tendance à modifier de manière asynchrone ces composants modularisés, la consistance de toute l'ontologie peut être affectée.

Changement de l'annotation sémantique

D'après [Prié et Garlatti, 2004], les termes de métadonnée ou d'annotation prennent bien en compte cette notion d'ajout d'information à une ressource, et on pourra a priori les utiliser indifféremment pour décrire les informations que le Web sémantique doit ajouter au Web pour le rendre plus utilisable par des machines. Une annotation sémantique est considérée comme une représentation formelle d'un contenu, exprimée à l'aide de concepts, relations et instances décrits dans une ontologie, et reliée à la ressource documentaire source [Amardeilh,

2007]. Donc, les changements de l'annotation sémantique sont étroitement liés aux modifications de l'ontologie et ils ont souvent lieu à cause de disparités présentes au niveau de la représentation du langage (i.e. le langage RDF).

Nous allons examiner plusieurs types de changements dans la partie scénarios d'évolution pour l'annotation sémantique (c.f. section 2.2.2)

Changement de la ressource

Pour réaliser le Web sémantique, il est nécessaire d'associer aux ressources du Web des informations exploitables par des agents logiciels afin de favoriser l'exploitation de ces ressources. Associer une information exploitable à une ressource signifie deux choses essentielles [Prié et Garlatti, 2004] :

- Cette information doit être structurée et descriptive pour décrire la ressource ;

- La ressource en question doit exister et pouvoir être exploitée sur le Web indépendamment des informations qui lui sont associées dans le cadre du Web sémantique : celles-ci sont utiles, mais non nécessaires pour accéder à la ressource, la page Web ou le service et l'utiliser.

Le cas du changement de la ressource le plus rencontré est la perte de référence d'une information sur une ressource. Les ressources sont identifiées par des identificateurs sous forme d'adresses URI [Miller et Manola, 2004], si on change les identificateurs de ressources sans mettre à jour les parties dépendantes, les informations annotant ces ressources pourraient générer des inconsistances.

D'autre part, la diversité des sources d'information distribuées et leur hétérogénéité sont une des principales difficultés rencontrées par les utilisateurs du Web aujourd'hui [Laublet et al., 2002]. Cette hétérogénéité peut provenir du format ou de la structure des sources (sources structurées : bases de données relationnelles, sources semi-structurées : documents XML, ou non structurées : textes), du mode d'accès et de requête, ou de l'hétérogénéité sémantique : entre les schémas conceptuels ou les ontologies implicites ou explicites sous-jacentes.

2.2.1.2 Interaction entre les changements des composants

Au cours de l'évolution d'un Web sémantique d'entreprise, deux types d'interaction entre changements des composants sont connus : d'une part sur les couples ontologie - annotation sémantique, et d'autre part sur les couples ressource - annotation sémantique (c.f. Figure 11).

Influence du changement de l'ontologie sur l'annotation sémantique

Les changements de l'ontologie de base doivent être propagés à toutes les ontologies dépendantes, aux instances ontologiques et aux logiciels/applications utilisant cette ontologie modifiée [Stojanovic et al., 2002b].

Dans notre approche, nous nous intéressons à la propagation de changements ontologiques à leurs instances et aussi vers les annotations sémantiques. Quand l'ontologie est modifiée, les instances doivent être changées de manière que les instances restent conformes à l'ontologie. Fondamentalement, si l'ontologie est modifiée alors ses instances doivent être transformées pour être conformes à la nouvelle ontologie obtenue après modification (nous l'appellerons ontologie modifiée). Cela signifie que l'adaptation continue des informations annotées à une nouvelle terminologie sémantique est nécessaire. Les changements de l'ontologie pourront influencer les annotations sémantiques, qui reposent sur le vocabulaire partagé de l'ontologie, et sur l'ontologie modifiée et générer des incompatibilités entre l'ontologie et ses annotations.

L'incompatibilité entre l'ontologie et les annotations peut se produire dans deux directions différentes : dans l'utilisation prospective et dans l'utilisation rétrospective [Klein, 2004]. L'utilisation prospective indique l'utilisation d'une nouvelle version de l'ontologie avec des sources de données conformes à une autre version plus ancienne tandis que l'utilisation rétrospective utilise une version ancienne de l'ontologie avec des sources de données conformes à une autre version plus récente.

Influence du changement de la ressource sur l'annotation sémantique

Les ressources sont identifiées par les identificateurs URI. Ces identificateurs sont aussi décrits dans les annotations sémantiques décrivant ces ressources. A cause des besoins de changement au niveau des ressources (i.e. suppression, déplacement ou bien mise à jour des ressources), les identificateurs pourraient être modifiés. Cela entraîne éventuellement la perte de référence de ces identificateurs URI dans la base d'annotations.

2.2.2 Scénarios d'évolution

2.2.2.1 Scénario d'évolution de l'ontologie

Dans [Haase et al., 2005], les auteurs mentionnent les quatre cas d'utilisation principaux qui requièrent des méthodes de résolution des inconsistances pour les ontologies évolutives.

- Changer une ontologie initiale consistante entraîne potentiellement des inconsistances dans cette ontologie. Ce type de changement se produit

typiquement pour les ontologies dont on doit maintenir la consistance au cours de leur évolution.

- Réutiliser une ontologie qui est encore inconsistante. Ce cas se produit également lorsqu'on ne peut pas vérifier la consistance de la source de l'ontologie.

- Dans certains cas, la consistance de l'ontologie n'est pas garantie et les inconsistances ne peuvent pas être réparées mais on veut quand même faire des requêtes sur cette ontologie. C'est le cas typique dans lequel le niveau de modèle (schéma) et le niveau d'instance de l'ontologie ont évolué séparément sans synchronisation des changements.

- Remplacer une version de l'ontologie par une autre version inconsistante.

Les cas d'utilisation mentionnés ci-dessus peuvent être considérés comme des scénarios d'évolution de l'ontologie. En effet, le scénario le plus typique, que la plupart des outils actuels supportent, est la modification de l'ontologie selon les requêtes de changements de l'utilisateur. Il y a également d'autres scénarios d'évolution pour les ontologies évolutives lorsqu'on fait des opérations de fusion d'ontologie ou d'alignement d'ontologies [Bruijn et al., 2004].

2.2.2.2 Scénario d'évolution de l'annotation sémantique

Nous avons présenté les causes/scénarios d'évolution de l'ontologie. Dans cette section nous examinons certains scénarios qui peuvent affecter la consistance de l'annotation sémantique [Luong et Dieng-Kuntz, 2006].

- *Scénario 1 :* Le fournisseur de l'ontologie fait des modifications sur son ontologie de base. A cause de ces changements ontologiques, les annotations peuvent être affectées entraînant un état inconsistant.

- *Scénario 2 :* L'utilisateur change ses annotations lui-même sans référer à l'ontologie de base utilisée par ces annotations. Les annotations modifiées peuvent être inconsistantes avec l'ontologie de base correspondante.

- *Scénario 3 :* L'utilisateur importe des annotations à partir d'autres sources. Ces annotations importées et les anciennes annotations reposent sur une même ontologie. Il pourrait y avoir des annotations redondantes.

- *Scénario 4 :* L'utilisateur effectue une migration de la base d'annotations vers un autre formalisme de représentation (par exemple

la migration de RDFS vers OWL). Il pourrait y avoir des inconsistances de syntaxe sur les annotations.

Cependant, le premier scénario dans lequel les changements de l'ontologie de base peuvent affecter l'état consistant des annotations sémantiques qui reposent sur les termes définis dans l'ontologie de base semble être le plus fréquent dans la réalité.

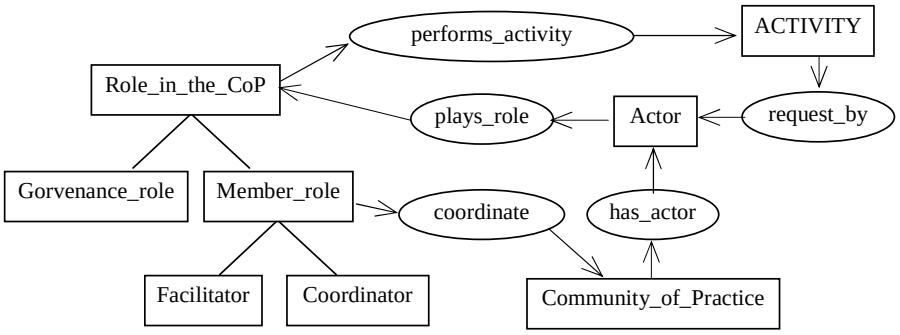


Figure 12 - Un extrait de l'ontologie O'CoP avant les changements

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <rdf:RDF
3.      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4.      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
5.      xmlns:owl="http://www.w3.org/2002/07/owl#"
6.      xmlns:xalan="http://xml.apache.org/xalan"
7.      xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
8.      xmlns:oc="http://www.inria.fr/acacia/ontocreator#"
9.      xml:base="http://www.inria.fr/acacia/ontocreator#">
10.
11. <oc:Actor rdf:about="http://www.inria.fr/acacia/ontocreator#WI-ACT_1"/>
12. <oc:Community_of_practice
13.     rdf:about="http://www.inria.fr/acacia/ontocreator#WI-CoP1"/>
14. <oc:ACTIVITY rdf:about="http://www.inria.fr/acacia/ontocreator#WI-AC2"/>

15. <oc:Member_role          rdf:about="http://www.inria.fr/acacia/ontocreator#
    Memrole1">
16.     <oc:performs_actvity
17.     rdf:resource="http://www.inria.fr/acacia/ontocreator#WI-AC2"/>
18. </oc:Member_role>

19. <oc:Coordinator      rdf:about="http://www.inria.fr/acacia/ontocreator#WI-
    Coor1">
20.     <oc:coordinate
21.     rdf:resource="http://www.inria.fr/acacia/ontocreator#WI-CoP1"/>
22. </oc:Coordinator>

23. <oc:Facilitator      rdf:about="http://www.inria.fr/acacia/ontocreator#WI-
    Faci1">
24.     <oc:coordinate
25.     rdf:resource="http://www.inria.fr/acacia/ontocreator#WI-CoP1"/>
26. </oc:Facilitator>

27. <oc:ACTIVITY         rdf:about="http://www.inria.fr/acacia/ontocreator#WI-
    AC_tmp1">
28.     <oc:request_by
29.     rdf:resource="http://www.inria.fr/acacia/ontocreator#WI-ACT_1"/>
30. </oc:ACTIVITY>
31. </rdf:RDF>
```

Figure 13 - Exemple d'une annotation basée sur l'ontologie O'CoP

Pour illustrer l'évolution de l'ontologie et de ses annotations sémantiques correspondantes dans le premier scénario, nous examinons un extrait d'une ontologie évolutive O'CoP [Tifous et Dieng-Kuntz, 2007] qui sera modifiée par certains changements après une mise à jour. Nous reposons sur le langage RDF(S) présenté dans [Klyne et Carroll, 2004] [Miller et Manola, 2004] pour modéliser l'ontologie et décrire un triplet `(s p v .)` en RDF dans l'annotation. Ce triplet représente une déclaration sur la ressource qui peut être exprimée comme *"un sujet* `s` *a une propriété* `p` *dont la valeur est* `v`*"*.

Examinons cet extrait de l'ontologie O'CoP (c.f. Figure 12) contenant le concept `Actor`, ce concept étant le domaine de la propriété `plays_role` et le co-domaine des propriétés `has_actor` et `request_by`. Le concept `Role_in_the_CoP` est le concept père des sous-concepts `Governance_role` et `Member_role`. Le concept `Memeber_role`, ayant deux sous-concepts `Facilitator` et `Coordinator`, est le domaine de la propriété `coordinate`. Les concepts `ACTIVITY` et `Community_of_Practice` sont les co-domaines d'une manière équivalente des propriétés `performs_activity` et `coordinate`. D'autre part, nous avons aussi les triplets suivants dans une annotation sémantique basée sur cette partie de l'ontologie (c.f. Figure 13).

Supposons que cet extrait de l'ontologie O'CoP soit modifié en supprimant le concept `Member_role`, ses deux sous-concepts sont reconnectés au concept `Role_in_the_CoP` et la propriété `coordinate` reçoit désormais le concept `Coordinator` comme son domaine. En plus, le concept `ACTIVITY` est renommé en `Activity`. Après avoir appliqué ces changements, nous obtenons une nouvelle version de l'ontologie O'CoP (c.f. Figure 14) dans laquelle certains éléments ont été changés par rapport à l'ancienne version. Certains triplets deviennent maintenant inconsistants (c.f. lignes en gras, Figure 13) du fait de la perte des liens référence vers les termes correspondants dans l'ontologie avant sa modification. Nous allons analyser les causes et les solutions pour corriger ces triplets inconsistants dans les chapitres suivants.

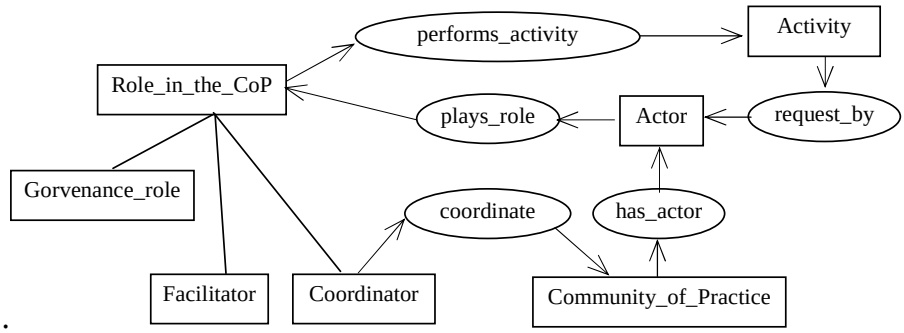


Figure 14 - Un extrait de l'ontologie O'CoP après les changements

2.3 Aspects importants de l'évolution de l'ontologie

Dans cette section, nous examinons les aspects importants qui sont mentionnés dans les recherches existantes sur l'évolution de l'ontologie. Ces aspects concernent aussi le sujet que nous traitons dans cette dissertation. Nous présentons les aspects suivants : processus d'évolution, représentation des changements ontologiques, détection des changements et niveaux de consistances, et nous faisons également une comparaison entre les recherches concernées et notre approche.

2.3.1 Processus d'évolution

Les auteurs dans [Stojanovic et al., 2002a] présentent un processus d'évolution de l'ontologie en six phases (1) capture du changement, (2) représentation du changement, (3) sémantique du changement, (4) propagation du changement, (5) implémentation du changement et (6) validation du changement. Dans ce processus, les deux phases les plus importantes sont (i) la sémantique de changement qui permet de résoudre des changements de l'ontologie d'une manière systématique en assurant la consistance de toute l'ontologie et (ii) la propagation du changement qui assure la consistance des parties dépendantes après avoir réalisé une mise à jour de l'ontologie. Ces parties dépendantes peuvent inclure des ontologies dépendantes, des instances ainsi que des programmes d'application utilisant l'ontologie de base modifiée.

Un autre processus similaire est introduit dans [Liang et al., 2006]. Les changements de l'ontologie entre deux versions sont d'abord capturés manuellement en utilisant l'outil Protégé [Noy et Musen, 2004] et sont représentés dans une ontologie des logs LogOntology. Ensuite, les requêtes soumises par l'application sont analysées pour vérifier la correspondance de chaque entité de la requête avec LogOntology. Toutes les entités correspondantes trouvées, qui sont mises à jour dans la nouvelle version, seront remplacées par les éléments équivalents de la nouvelle version. Enfin, les informations du processus de mise à jour ainsi que les informations sur les nouveaux éléments sont renvoyées aux utilisateurs.

Dans [Klein et Noy, 2003] et [Klein, 2004], les auteurs définissent un *framework* pour l'évolution avec un processus de gestion des versions de l'ontologie. Les étapes principales de ce processus sont : (1) l'identification de deux versions de l'ontologie appropriées, (2) la création de la spécification du changement entre deux versions, et (3) la génération de nouvelles connaissances sur le lien entre les deux versions.

Une des recherches les plus récentes [Rogozan et al., 2005] présente l'utilisation des ontologies évolutives dans un système de téléapprentissage. Les auteurs définissent l'évolution de l'ontologie comme le processus de changement

de la version précédente V_N de l'ontologie à une nouvelle version V_{N+1}, afin de rendre compte des modifications dans le domaine de l'ontologie, dans sa conceptualisation ou dans son usage. Comme les approches précédentes, leur processus contient des étapes principales telles que l'identification et l'édition des changements, la vérification et l'implémentation des changements, l'analyse de la compatibilité entre des versions V_N et V_{N+1}.

Notre travail présenté dans ce manuscrit ne se concentre pas principalement sur le processus d'évolution de l'ontologie car nous nous intéressons au processus de propagation de changement de l'ontologie vers les annotations sémantiques reposant sur les concepts et relations de cette ontologie. Cependant, nos travaux sur l'évolution de l'ontologie sont intégrés dans une application basée sur le web : ECCO, un éditeur d'ontologie développé par notre équipe. Cet outil d'édition supporte le processus de représentation de l'ontologie, de modification et de résolution des inconsistances générées par des changements ontologiques. Dans le processus d'évolution, cet outil va créer automatiquement un journal d'évolution qui capture tous les types de changements exécutés au cours de modification de l'ontologie. Ce journal d'évolution est utilisé ensuite dans notre travail sur l'évolution de l'annotation permettant d'analyser et réparer des annotations inconsistantes.

2.3.2 Représentation des changements

Afin de faciliter l'évolution de l'ontologie dans un environnement dynamique où les utilisateurs ne sont pas tous experts dans la construction d'ontologie, il est important de formaliser les opérations qui pourraient changer la base de connaissances en vue d'obtenir une sémantique explicite pour chaque opération. Les changements de l'ontologie doivent être identifiés et représentés dans un format approprié.

Les changements de l'ontologie peuvent aussi être représentés pour les différents niveaux de granularité. La plupart des recherches existantes [Plessers et Troyer, 2005], [Klein, 2004], [Stojanovic, 2004] classent les changements en deux types principaux : (i) *changement élémentaire (ou atomique)* traitant une seule entité de l'ontologie et pouvant être vu comme une modification isolée dans l'ontologie, (ii) *changement composite* traitant plusieurs entités de l'ontologie et considéré comme une composition de plusieurs changements élémentaires. Des exemples de changements élémentaires de l'ontologie simples sont les opérations `Supprimer` ou `Ajouter` un concept (ou une propriété). Le changement composite `Déplacer` permet de déplacer un concept vers une autre position dans la hiérarchie de concepts. Un intérêt du changement composite est qu'il permet à

l'ontologiste de formuler ses requêtes de changement au niveau d'abstraction le plus élevé, au lieu de les exprimer par différents changements élémentaires. Cependant, d'après [Stojanovic, 2004] une composition pure de changements élémentaires n'est pas assez puissante pour exprimer une grande classe des transformations. Par conséquent, le niveau d'abstraction plus haut des changements d'ontologie inclut les changements complexes. Les auteurs définissent *un changement complexe* comme pouvant être décomposé en n'importe quelle combinaison contenant au moins deux changements, l'un élémentaire et l'autre composite, de l'ontologie. D'autre part, les changements de l'ontologie sont décrits non seulement selon le niveau de granularité mais aussi selon les langages de définition des changements. Dans [Plessers et Troyer, 2005], les auteurs proposent un langage permettant de spécifier les définitions de changement. A ce propos, [Klein, 2004] présente un langage de spécification des changements qui repose sur la syntaxe basée en RDF pour spécifier des opérations de changement.

Les changements sont représentés selon différents niveaux d'abstraction, mais comment sont ils conservés et traités pendant le processus d'évolution ? [Stojanovic, 2004] propose d'utiliser un journal d'évolution qui représente des changements entre deux versions d'une ontologie. Ce journal conserve l'historique des modifications de l'ontologie sous forme d'une chaîne de changements ontologiques exécutés. Les changements ontologiques sont exprimés en utilisant les termes définis dans une ontologie d'évolution dédiée à la représentation des types de changements de l'ontologie. Une autre approche similaire [Plessers, 2006] utilise deux types de journal : (i) le journal de version pour sauvegarder les différentes versions des concepts de l'ontologie et (ii) le journal d'évolution pour sauvegarder les interprétations de ces versions en termes de changements. Dans [Klein, 2004], les changements entre deux versions de l'ontologie sont représentés sous forme d'un fichier de journal qui décrit un ensemble de transformations et de relations conceptuelles. Actuellement, plusieurs outils d'édition d'ontologie tels que Protégé [30], OntoEdit [31] fournissent les fichiers journaux permettant de conserver les traces de changements exécutés au cours de l'évolution.

[30] http ://protege.stanford.edu/

[31] http ://www.ontoknowledge.org/tools/ontoedit.shtml

Dans notre approche, nous distiguons deux niveaux de consistance différents du modèle de l'ontologie, i.e. la consistance struturelle et la consistance logique. La consistance structurelle concerne les contraintes de consistance définies pour un modèle de l'ontologie en assurant une bonne organisation des éléments de l'ontologie (e.g. le concept, la propriété) au niveau de structure. La consistance logique s'intéressse à savoir si les éléments de l'ontologie restent encore "semantiquement corrects". Le travail principal de cette thèse comporte la détection et la résolution des inconsistances de l'annotation sémantique qui sont reliées étroitement à la consistance structurelle. Nous nous donc intéressons à l'évolution de l'ontologie au niveau de la consistance structurelle et nous ne traitons pas la consistance logique. En plus, nous nous focalisons sur les changements ontologiques qui peuvent affecter la consistance de l'annotation, par conséquent une classification de changement au niveau élémentaire et composite est suffisante pour exprimer notre besoin de représentation de changement. D'autre part, nous distinguons dans notre approche les changements ontologiques de (i) correction obligatoire qui influencent la consistance des annotations sémantiques et (ii) correction facultative qui pourraient ne pas affecter la consistance des annotations sémantiques. Dans le chapitre 4, nous introduirons d'une manière détaillée la construction d'une ontologie d'évolution qui décrit les types de changements utilisés. Pour garder la trace d'évolution au cours de modifications de l'ontologie, nous utilisons un journal sémantique généré automatiquement par l'outil d'édition d'ontologie ECCO. Ce journal d'évolution contient des éléments décrivant les changements exécutés qui sont formalisés d'une manière formelle grâce aux concepts et relations de l'ontologie d'évolution.

2.3.3 Détection des changements ontologiques

Les changements exécutés pendant l'évolution de l'ontologie dans le projet KAON [Maedche et al., 2003] [Stojanovic, 2004] sont enregistrés dans un journal permettant de tracer l'historique et les éléments ontologiques affectés par ces changements. Cependant, seuls les changements élémentaires sont sauvegardés dans ce journal selon l'ordre de leur exécution (par exemple les changements du type `AddConcept` ou `DeleteConcept`). Par conséquent, la trace des changements du journal ne facilite pas le travail pour rendre compte des changements complexes (par exemple les changements du type `MergeConcept`), cela empêche l'analyse des effets des changements sur la compatibilité entre les versions de l'ontologie ou sur la relation sémantique entre leurs entités. Pour résumer, cette approche détecte des changements exécutés grâce à la trace des changements entre deux versions. De la même manière, [Plessers, 2006] utilise le journal de versions pour découvrir des changements conservés en exécutant des requêtes temporelles sur ce journal.

Cependant, dans le contexte distribué et évolué du Web sémantique, on ne peut pas toujours garder d'une manière complète des versions de l'ontologie ainsi que la trace de changement entre ces versions. Pour identifier des changements ontologiques dans ce scénario, [Klein et al., 2002] présente l'outil OntoView qui permet de comparer les versions d'une ontologie au niveau structurel afin de découvrir les définitions modifiées d'une version à l'autre. Les auteurs présentent un mécanisme de détection basé sur des règles et des heuristiques pour détecter des opérations complexes de changement entre deux versions de l'ontologie (V_{OLD} et V_{NEW}). Bien que leur approche soit applicable dans certains cas spécifiques, généralement elle a des limitations pour un processus de détection imprécis et imprévisible.

Un autre outil PromptDiff, qui ne nécessite aucune information supplémentaire sur la trace d'évolution de l'ontologie, est présenté dans [Noy et Musen, 2004]. Cet outil effectue aussi une comparaison entre les versions d'une ontologie en mettant en correspondance les entités ontologiques d'une version avec celles d'une autre version. Cependant, PromptDiff n'aborde pas une technique d'identification des changements. Il ne fournit aucune information au sujet des changements ontologiques et indique seulement si les entités ont été modifiées ou non.

Dans notre travail, les changements exécutés et les informations du processus d'évolution sont capturés dans un journal de trace de changements (ou journal d'évolution) sous forme d'une annotation, cela nous permet de la manipuler facilement grâce à sa sémantique exprimée par une ontologie d'évolution. En découvrant les informations capturés dans ce fichier journal, nous pouvons déterminer quels sont les changements ontologiques qui ont été effectués. Quels éléments seront affectés à cause de ces changements? Ceci est le premier cas d'évolution de l'ontologie avec une trace de changement qui est présenté dans le chapitre 4. Pour le deuxième cas d'évolution sans la trace de changements entre des versions de l'ontologie, les recherches actuelles ne donnent pas une solution pour déterminer les changements de l'ontologie effectués à cause de la manque d'information sur le processus d'évolution. Au lieu de déterminer quels changements ont été effectués, notre approche vise à proposer à l'utilisateur les meilleures manières de corriger les annotations sémantiques inconsistantes. Nous développons un mécanisme pour choisir le "meilleur" élément dans la nouvelle version de l'ontologie pour remplacer celui qui devient obsolète dans l'annotation. Nous présentons ce travail dans le chapitre 5.

2.3.4 Vérification et résolution des inconsistances

Après la modification de l'ontologie, il est nécessare de vérifier la consistance de tous les éléments de l'ontologie et de ses parties dépendantes. Si

les inconsistances ont lieu au cours de l'évolution, elles doivent être ensuite résolues afin de garantir la consistance de toute l'ontologie.

Dans l'approche de l'évolution proposée par [Stojanovic, 2004], la vérification de consistance au niveau de la structure est basée sur l'ensemble des invariants construits qui doivent être satisfaits pour chaque ontologie tandis que les inconsistances sont résolues en appliquant des règles. Suivant l'idée dans la phase de la sémantique de changements qui assure la consistance basée sur l'adaptation des invariants et des règles [Banerjee et al., 1987], Stojanovic et al. définissent le modèle de consistance de l'ontologie M comme une union de 16 invariants et de certaines contraintes définies par l'utilisateur qui sont appliqués dans le langage d'ontologie de KAON. Les auteurs ont adopté les deux approches : approche procédurale et approche déclarative de l'évolution du schéma de la base de données à l'évolution de l'ontologie pour résoudre des changements induits d'une manière systématique, assurant la consistance de toute l'ontologie. D'autre part, comme une inconsistance peut être généralement résolue de différentes manières, les auteurs ont proposé la notion de stratégies d'évolution qui permettent à l'ontologiste de choisir les solutions les plus convenables pour leur résolution d'inconsistance.

[Rogozan et al., 2005] propose de construire une méthode pour assurer un référencement sémantique évolutif (d'après leur contexte de recherche, l'ontologie est utilisée comme référentiel sémantique pour les éléments pédagogiques du système d'apprentissage) réalisé par la modification et la mise à jour des liens de références, c'est-à-dire les liens qui relient un objet à sa référence dans un espace ontologique, pour les objets référencés à l'aide des entités d'une ontologie évolutive.

Cependant, les recherches ci-dessus ne traitent que la consistance au niveau de la structure. Les auteurs dans [Plessers, 2006] proposent une solution de maintenance de la consistance logique. Ils ont défini un algorithme pour déterminer l'ensemble d'axiomes causant une contradiction logique. Les auteurs présentent aussi des règles qui guident l'ontologiste dans son choix d'axiomes à changer pour résoudre la contradiction détectée. D'autre part, les auteurs de [Haase et Stojanovic, 2005], [Haase et al., 2005] discutent les trois formes différentes de consistance suivantes : la consistance structurelle, la consistance logique et la consistance définie par l'utilisateur. Pour résoudre des inconsistances structurelles, ils présentent une approche basée sur des règles de réécriture qui permettent de transformer des axiomes en variante désirée en OWL. Pour résoudre des inconsistances logiques, ils présentent deux approches alternatives pour localiser une inconsistance d'une ontologie en OWL : (i) trouver une sous-ontologie consistante maximale et (ii) trouver une sous-ontologie inconsistante minimale.

Notre approche diffère des travaux traitant de la consistance au niveau logique. Dans le but de décrire notre approche sur l'évolution au niveau générique, nous nous intéressons à un modèle de représentation des connaissances au niveau structural, donc nous modélisons les ontologies et les annotations sémantiques dans le langage RDF(S) [Miller et Manola, 2004], le langage recommandé par le W3C. Dans notre équipe, le travail de résolution d'inconsistance est intégré dans l'éditeur d'ontologies ECCO. Nous établissons également un ensemble des stratégies de résolution pour l'ontologie qui guident l'ontologiste dans la modification et le choix de la solution la plus convenable pour chaque type de changement. Nous nous focalisons dans ce mémoire sur la résolution des inconsistances des annotations, générées à cause des changements ontologiques. Cependant, notre outil a une fonctionnalité de visualisation des différences entre les deux versions de l'ontologie, cela facilite la vérification de la consistance de l'ontologie et fournit plus d'information à l'utilisateur pour la phase de correction des inconsistances.

2.3.5 Comparaison et Gestion des versions de l'ontologie

La gestion des versions de l'ontologie fournit des mécanismes pour gérer et utiliser plusieurs versions modifiées ou extensibles de l'ontologie. Le but général d'une méthodologie de gestion des versions est de gérer l'interopérabilité. Dans le contexte du Web sémantique, l'interopérabilité peut être définie comme la capacité d'utiliser différentes versions de l'ontologie avec les sources de données sans avoir de conflits.

Les recherches existantes sur la gestion des versions de l'ontologie s'intéressent à deux problèmes : (1) identification des versions de l'ontologie dans des environnements distribués tels que le Web sémantique [Klein and Fensel, 2001] et (2) spécification explicite du journal de changements entre des versions [Heflin et Hendler, 2000]. L'identification des versions assure une référence non ambigüe à la définition prévue pour l'utilisation d'un concept ou d'une propriété. Les spécifications de changement sont des descriptions explicites de l'évolution entre deux versions d'une ontologie. Le format de spécification de changement est normalement fixé par les opérations du changement de l'ontologie et le format de représentation du changement par exemple dans le *framework* WonderWeb[32]. Cependant, le journal de changement entre des versions n'est pas toujours disponible pour les ontologies distribuées, certains outils ont vaincu cet obstacle en fournissant une manière automatique de comparer différentes versions basées sur la sémantique codée en leur structure lorsque le journal de changements n'est pas disponible [Noy et Musen, 2002] [Klein et al., 2002].

[32] http ://wonderweb.semanticweb.org/

Pour l'aspect lié à la gestion des versions, notre approche permet de travailler avec plusieurs versions différentes de l'ontologie. Les utilisateurs peuvent choisir chaque fois deux versions différentes pour comparer les différences d'une manière visuelle. S'il n'y pas de trace de changements, notre approche peut lister d'une manière simple les concepts et les propriétés qui différent entre deux versions de l'ontologie. Par exemple, les concepts qui existent dans la première version mais n'existent pas dans la deuxième version et vice-versa. Dans le cas où on possède l'information sur les changements effectués dans la trace d'évolution, notre approche peut fournir plus d'information décrivant la différence entre deux versions ontologiques à propos de chaque changement effectué. D'autre part, les versions modifiées de l'annotation sont aussi sauvegardées après chaque mise à jour selon le choix de l'utilisateur.

2.3.6 Discussion

Au sujet de l'évolution, les recherches récentes se focalisent principalement sur deux grandes approches : (i) évolution de l'ontologie qui permet l'accès aux données via la version de l'ontologie la plus récente et (ii) gestion des versions de l'ontologie qui permet l'accès aux données par différentes versions de l'ontologie. Les recherches actuelles ont abordé également le problème de consistance d'une ontologie à deux niveaux d'abstraction : structurel et logique.

Nous avons examiné dans cette section certains aspects importants de l'évolution de l'ontologie que les outils actuels supportent. Ces aspects sont aussi intéressants dans notre approche mais ils ne sont pas encore bien traités entièrement. Dans le tableau suivant (c.f. Tableau 1), nous faisons un résumé des fonctionnalités principales ainsi que de la capacité de support (i.e. Non/Oui= non/oui support, Partiel=support partiel) de chaque approche pour résoudre les aspects de l'évolution abordés.

Tableau 1 - Comparaison de certaines approches sur l'évolution de l'ontologie

Approches / **Fonctionalités**	**[Noy et Musen, 2002]**	**[Stojanvic, 2004]**	**[Klein, 2004]**	**[Rogozan et Paquette, 2005]**	**[Plessers, 2006]**	**[Flouris, 2006]**
Outil/Prototype	**Prompt-Diff** **Protégé**	**KAON Tool Suite**	**Onto-View** **Wonder-Web**	**Onto-Analyseur**	**Plugs-in Protégé** **FaCT++**	**AGM**

Processus d'évolution	Non	Oui	Non	Oui	Oui	Partiel
Représentation du changement	Partiel	Oui	Oui	Non	Oui	Oui
Détection du changement	Oui	Non	Oui	Partiel	Partiel	Oui
Vérification de consistance	Non	Oui	Non	Oui	Oui	Oui
Résolution d'inconsistance	Non	Oui	Non	Oui	Oui	Oui
Comparaison des versions	Oui	Non	Oui	Partiel	Non	Non
Gestion des versions	Non	Non	Non	Partiel	Non	Non
Propagation du changement aux ontologies dépendantes	Non	Oui	Non	Non	Non	Non
Propagation du changement aux annotations sémantiques	Non	Non	Non	Oui	Non	Non

À travers ce tableau, nous réalisons que les recherches actuelles ne supportent pas entièrement les problèmes de détection du changement, de comparaison et de gestion des versions de l'ontologie. Particulièrement, les propagations des changements ontologiques aux ontologies dépendantes et aux annotations sémantiques sont rarement abordées et elles ne sont pas bien traitées.

2.4 Aspects importants de l'évolution de l'annotation sémantique

Les métadonnées ou les annotations sémantiques décrivant des ressources Web sont utilisées dans tous les systèmes d'indexation de documents Web, que ce soit des moteurs de recherche, des annuaires, des signets personnels, et de façon plus générale dans tous les systèmes de gestion d'information, ces derniers bénéficiant d'une vieille tradition dans le monde de la documentation et des bibliothèques (schéma d'indexation, utilisation de thésaurus, etc.) [Prié et Garlatti, 2004]. Cependant, la plupart des méthodes et des outils de l'annotation se

concentrent sur le problème de construction et particulièrement sur le processus de création des annotations à partir des thésaurus ou des documents textuels. Mais il est très rare qu'ils concernent des approches traitant le problème de l'évolution de l'annotation sémantique.

Puisque nous nous intéressons à la gestion des annotations sémantiques, nous allons donc examiner, dans cette section, certains aspects importants de l'évolution de l'annotation sémantique : l'analyse des effets des changements ontologiques sur l'annotation, la détection des annotations inconsistantes et la résolution des annotations inconsistantes.

2.4.1 Analyse des effets des changements ontologiques sur l'annotation

Très peu d'approches ont étudié les problèmes de la propagation des changements ontologiques aux annotations sémantiques. Dans (Stojanovic et al., 2002), est proposée la plate-forme CREAM pour résoudre l'évolution des métadonnées basée sur l'évolution de l'ontologie. Cependant, cette approche ne présente qu'une proposition d'une plate-forme pour permettre la consistance des descriptions des sources de connaissances en cas de changements de l'ontologie du domaine mais ne propose aucune technique pour la résoudre.

Un autre travail sur l'influence de l'évolution d'ontologie sur les métadonnées par les contraintes relationnelles d'un système de base de données est également présenté dans [Ceravolo et al., 2004]. Dans leur projet, ils ont implémenté un méta-modèle d'Entité-Relation (ER) pour l'information de RDF. Le modèle ER représente des relations structurées comme triplets en utilisant la syntaxe de logique prédicat : (<étiquette-arc>,<objet>,<valeur-propriété>). La collection de triplets forme une relation de base de données sauvegardant les méta-données du système. L'idée principale de cette approche [Ceravolo et al., 2004] est de construire une classification des opérations et des *triggers*. Les auteurs introduisent les opérations *sans-contexte* dont les effets ne sont pas influencés par le type de relation qui se produisent parmi des classes impliquées. Au contraire, les opérations *sensibles au contexte* sont celles qui produisent différents effets selon le type de relation de classes impliquées. Ensuite, les auteurs considèrent l'événement de maintenance de l'ontologie comme des *triggers* de bases de données et la réalisation du changement de l'ontologie comme l'exécution des *triggers*. Les *triggers* de base de données sont une technique bien connue pour définir un ensemble d'actions tel qu'une insertion, suppression ou mise à jour.

[Rogozan et Paquette, 2005] présente un mécanisme d'analyse des versions de l'ontologie dans lequel ils proposent une méthode qui propage les changements ontologiques, effectués pour passer de V_N à V_{N+1}, dans les objets référencés par l'ontologie. La propagation des changements dans les objets référencés signifie la

modification des liens de référence, un lien de référence étant le chemin qui relie un objet (i.e. une ressource annotée) à une entité ontologique utilisée comme référence sémantique.

Dans le processus d'évolution de l'annotation sémantique, notre approche permet de connaître les annotations qui deviennent inconsistantes après les modifications de l'ontologie de base. D'une manière plus précise, on peut trouver des annotations inconsistantes dues à un changement ontologique quelconque. Nous pouvons également analyser quels sont les triplets inconsistants, ainsi que leur provenance ou annotation d'origine. Les interfaces de visualisation dans notre outil CoSWEM, qui implémentent ces fonctions vont faciliter la tâche utilisateur d'analyse et permettre de mieux comprendre les changements de l'ontologie sur la base d'annotations concernée.

2.4.2 Détection des annotations inconsistantes

Une question est encore ouverte au sujet de la détection d'inconsistance dans l'évolution de l'annotation sémantique : comment peut-on savoir quelles sont les inconsistances générées dans les annotations après des changements de l'ontologie et de l'annotation elle-même ? Les recherches actuelles ne donnent pas encore de solution pertinente à cette question. Le travail qui concerne le plus ce point [Rogozan et Paquette, 2005] n'aborde pas non plus ce problème bien qu'il propose une solution de modification des références sémantiques dans l'annotation.

Notre approche s'intéresse à la phase de détection des inconsistances produites dans les annotations. Nous proposons deux approches pour détecter les inconsistances générées de l'annotation sémantique après avoir changé l'ontologie de référence : (i) *l'approche procédurale* est appliquée dans le cas où il existe un journal de trace qui capture des changements ontologiques exécutés et (ii) *l'approche basée sur des règles*, quant à elles, est dédiée aux cas où l'on ne garde plus la trace de changement entre les versions de l'ontologie (voir le chapitre 5). D'autre part, notre outil a une fonctionnalité de visualisation des différences entre deux versions de l'ontologie et les annotations basées sur ces différences, cela donnera plus d'information à l'utilisateur sur les annotations inconsistantes détectées.

2.4.3 Résolution des annotations inconsistantes

Après avoir détecté des annotations inconsistantes, comment peut-on les résoudre et mettre à jour la base de connaissances ? Ce problème est encore ouvert pour plusieurs recherches existantes.

Le travail de [Ceravolo et al., 2004] n'est pas très lié à l'annotation sémantique mais il repose sur l'influence de l'évolution d'ontologie sur les métadonnées par les contraintes relationnelles d'un système de base de données. Selon cette approche, une base de données relationnelle est utilisée pour stocker les descriptions en RDF(S) représentant les assertions basées sur l'ontologie et la structure d'ontologie elle même. La maintenance d'ontologie peut être contrôlée en utilisant des triggers de base de données pour modifier automatiquement des domaines ou des co-domaines de propriété dans les assertions stockées.

[Rogozan et Paquette, 2005] présente un contexte particulier dans lequel l'ontologie est utilisée comme référentiel sémantique pour les éléments pédagogiques d'un système d'apprentissage sur le Web sémantique. Pour préserver ce rôle de l'ontologie, ils proposent une méthode de propagation des changements ontologiques dans les objets référencés par l'ontologie. La propagation des changements dans les objets référencés signifie la modification des liens de références, un lien de référence étant le chemin qui relie un objet à une entité ontologique utilisée comme référence sémantique. Selon cette approche, les auteurs définissent un UKI (Uniform Knowledge Identifier) exprimant le lien de référence qui relie un objet à un de ses repères sémantiques dans un espace de connaissances. Ensuite, ils proposent de changer cet UKI pour conserver le rôle d'une ontologie évolutive utilisée comme référentiel sémantique (i.e. changer le nom UKI "http://www.w3.org/ActeursTA_3.0#Formateur" par "http://www.w3.org/ActeursTA_3.0#Formateur_Etudiant" s'il existe dans l'ontologie une modification de type renommage du concept Formateur à Formateur_Etudiant).

Pour la résolution des annotations inconsistantes détectées, nous avons proposé deux méthodes : méthode *automatique* ou *semi-automatique*. Dans le cas où on connaît comment les changements de l'ontologie ont été résolus et les stratégies de résolution appliquées pour l'ontologie, nous pouvons utiliser des règles de correction afin de réaliser automatiquement la stratégie de résolution choisie pour l'annotation sémantique qui a été construite respectivement pour chaque stratégie de l'ontologie. D'autre part, si nous n'avons aucune information sur l'évolution de l'ontologie, nous pouvons réaliser la tâche de résolution des annotations inconsistantes d'une manière semi-automatique avec l'aide de l'utilisateur pour choisir une opération qui lui convient. Correspondant à chaque changement réalisé, nous proposons des stratégies d'évolution permettant de résoudre des inconsistances. Les utilisateurs peuvent alors choisir l'opération à appliquer qui leur semble la plus convenable à leurs besoins. Nous détaillons ces deux approches dans les chapitres 4 et 5.

2.4.4 Discussion

Nous choisissons les trois approches de recherches existantes (c.f. Tableau 2) qui sont les plus liées avec les travaux sur l'évolution de l'annotation et sont aussi proches de notre recherche sur la gestion des annotations inconsistantes, pour les comparer.

Tableau 2 – Comparaison de certaines approches sur l'évolution de l'annotation

Approches / **Fonctionalités**	**[Stojanvic, 2004]**	**[Rogozan et Paquette, 2005]**	**[Liang, 2006]**
Outils/Prototype	**CREAM** (proposition)	**UKI-Modificateur** (proposition)	**CRM** (proposition)
Processus d'évolution	Oui	Oui	Oui
Analyse des effets de changements ontologiques	Non	Oui	Partiel
Détection des annotations inconsistantes	Non	Partiel	Non
Résolution des annotations inconsistantes	Non	Partiel	Non

Dans ce tableau, nous trouvons que les recherches sur la détection et la résolution des annotations inconsistantes ne sont pas bien traitées. Le travail de [Rogozan et Paquette, 2005] a mentionné ces problèmes mais il ne traite que partiellement la résolution des inconsistances, et leur résultat reste encore des propositions au niveau théorique. Par conséquent, nous nous focalisons dans notre recherche sur l'évolution de l'annotation sémantique avec ces deux problèmes de détection et de résolution des annotations inconsistantes.

2.5 Conclusion

Les connaissances organisationnelles sont considérées comme un capital important de toute organisation. De plus en plus, les organisations exploitent un système de gestion des connaissances afin de faciliter l'accès, le partage et la réutilisation des connaissances organisationnelles. Nous avons présenté dans ce chapitre l'approche du Web sémantique d'entreprise proposée par l'équipe ACACIA comme une solution de gestion des connaissances pour les entreprises ou organisations dans le contexte hétérogène et distribué du Web sémantique.

Nous avons fait un bilan des causes possibles qui entraînent l'évolution d'un tel Web sémantique d'entreprise et aussi de certains scénarios d'évolution de l'ontologie et de l'annotation sémantique. Nous soulignons le scénario, fréquent dans le monde réel, dans lequel les modifications de l'ontologie de base affectent la consistance des annotations sémantiques reposant sur cette ontologie. D'autre part, nous avons également fait un résumé des aspects importants concernant l'évolution de l'ontologie et de l'annotation sémantique, particulièrement les aspects qui sont traités dans ce manuscrit.

À travers ces recherches similaires, on trouve que les méthodes et les outils existants adressent principalement le problème de construction du système de gestion des connaissances mais très peu de recherches mentionnent le problème de la gestion de l'évolution qui n'est pas encore bien résolu jusqu'à présent (Lindgren et al., 2002). Nous détaillons dans le chapitre suivant notre proposition d'un système de gestion de l'évolution, CoSWEM (Corporate Semantic Web Evolution Management), permettant de gérer l'évolution du Web sémantique d'entreprise et particulèrement de gérer les annotations sémantiques inconsistantes dans le contexte des modifications de l'ontologie.

Partie II

Vers le système de gestion de l'évolution CoSWEM

Chapitre 3

CoSWEM – un système de gestion de l'évolution du WSE

Pour un système de gestion des connaissances basé sur le Web sémantique, les ontologies et les annotations sémantiques sont deux composants importants qui facilitent le processus de formalisation et de manipulation des connaissances. Dans l'environnement dynamique et distribué du Web sémantique, les ontologies et les annotations pourraient être changées pour s'adapter à l'évolution de l'organisation concernée. Ces changements peuvent donc entraîner des inconsistances affectant le fonctionnement des applications du Web sémantique qui sont basées normalement sur des sources d'information hétérogènes et fortement distribuées. Nous avons donc besoin d'un mécanisme pour contrôler les changements internes et externes du système de gestion des connaissances afin d'assurer la consistance de tout le système. Dans ce chapitre, nous évoquons les besoins qui ont motivé l'élaboration du système CoSWEM (Corporate Semantic Web Evolution Management). Ce système a pour but de gérer l'évolution du Web sémantique d'entreprise, particulièrement sur les deux composants importants, i.e. l'ontologie et l'annotation sémantique. Nous présentons également certains outils de support qui sont utilisés dans CoSWEM pour effectuer des tâches spécifiques.

3.1 Motivations

Dans les systèmes de gestion des connaissances, l'évolution est une dimension importante qui pourrait mener à des échecs imprévus dans différents niveaux du système [Lindgren et al., 2000]. L'évolution de ces systèmes est due à plusieurs raisons : le monde change, les exigences de métiers changent, la stratégie change ou le contexte change. Ce problème d'évolution doit être bien géré pour que le système puisse s'adapter aux nouveaux changements. Cependant, la plupart des recherches actuelles sur les systèmes de gestion des connaissances se focalisent sur la construction du système mais très peu sur la gestion de l'évolution du système.

Nous nous intéressons dans ce manuscrit au problème d'évolution dans un Web sémantique d'entreprise, particulièrement à l'évolution sur ses composants principaux, i.e. l'ontologie et l'annotation sémantique. Nous avons ainsi développé un système, CoSWEM (Corporate Semantic Web Evolution Management), permetttant de gérer l'évolution de ces composants dans les applications du Web sémantique. Les besoins analysés ci-dessous nous renforcent dans l'idée de développer un tel système de gestion de l'évolution :

- *Besoin de gérer l'évolution de l'ontologie :* Le facteur principal de changement est l'ontologie car le domaine de l'ontologie change rapidement ainsi que les besoins de mettre à jour l'ontologie (e.g. un concept est supprimé, un lien hiérarchique des concepts est modifié, etc.). Au cours de l'évolution, nous nous intéressons aux changements de l'ontologie ? Comment savoir les changements sont exécutés ? Comment la propagation des changements ontologiques est réalisée ? etc.

- *Besoin de gérer l'évolution de l'annotation sémantique :* Les annotations sémantiques reposent sur les concepts et les annotations définis dans l'ontologie de domaine. Elles pourront évoluer suite à des modifications de l'ontologie ou par elles-mêmes. D'autre part, les annotations sémantiques sont aussi affectées par les ressources qu'elles annotent. Nous avons besoin de déterminer quelles annotations et triplets seront affectés par les changements ontologiques ? Par quels changements de l'ontologie, les annotations sémantiques deviennent-elles inconsistantes ? Quels sont leurs types d'inconsistances ?

- *Besoin de mettre à jour les éléments inconsistants après modifications :* Les annotations sémantiques inconsistantes doivent être corrigées pour assurer la consistance de toute la base d'annotations. Comment sont-elles mises à jour ?

- *Besoin de consulter les informations parmi plusieurs versions :* Au cours de l'évolution, les ontologies et les annotations peuvent être sauvegardées en différentes versions. Cela nous permet de choisir les versions convenables sur lesquelles travailler selon notre besoin.

- *Besoin d'avoir un outil de visualisation :* Cet outil permettra à l'utilisateur de surveiller les différences entre les versions de l'ontologie, les changements exécutés entre ces versions, les annotations sémantiques inconsistantes à cause de ces changements, etc.

Ces principaux besoins du système CoSWEM sont rassemblés et développés dans les fonctions correspondantes. Dans les parties suivantes de ce chapitre, nous présentons l'architecture, les fonctions principales, les processus ainsi que les outils de support utilisés pour développer le système CoSWEM.

3.2 Architecture du système CoSWEM

3.2.1 Architecture

L'architecture du système de gestion de l'évolution CoSWEM se compose des composants principaux ci-dessous (c.f. Figure 15) :

- ***Composant d'Utilisateur :*** Nous distinguons deux types d'exigence au niveau des changements du côté du client qui peuvent influencer la consistance du système. Des changements d'exigences de métiers sont normalement demandés à l'entreprise ou aux utilisateurs intéressés par les aspects de métiers. Des changements techniques sont apportés par l'ontologiste ou l'annotateur qui comprend les aspects techniques du système. Les utilisateurs envoient la demande de changement au composant intermédiaire qui les notifie après la résolution des changements.

- ***Composant Intermédiaire :*** D'un point de vue général, tous les changements demandés sont prétraités dans cette couche intermédiaire. Tout d'abord, ils sont capturés et représentés dans un format approprié (changements élémentaires et composites). Ces changements sont alors classés selon le type des changements exécutés qui se sont produits dans différentes parties (ressource, ontologie ou annotation sémantique) et ils seront résolus dans le composant d'évolution. Cette couche a également une fonction de notification aux utilisateurs du résultat de la résolution des changements ainsi que des éléments qui ont été mis à jour. Dans le cas de l'évolution où on peut garder la trace des changements effectués entre les versions de l'ontologie, ces

changements sont représentés d'une manière formelle dans cette couche afin de les réutiliser pour les phases suivantes.

- ***Composant d'Evolution :*** Ce composant principal du système permet de résoudre des problèmes d'évolution générés dans chaque partie (i.e. ressource, ontologie et annotation sémantique). Nous divisons les changements d'évolution en trois types correspondant aux différentes parties d'un Web sémantique d'entreprise. Normalement, les changements des ressources et des ontologies impliqueraient des changements sur les annotations sémantiques concernées. Cependant, dans le cadre de notre recherche, nous nous concentrons principalement sur les changements des ontologies qui pourraient entraîner les modifications des annotations sémantiques. Au cours de l'évolution, nous construisons un journal d'évolution permettant de capturer tous les changements effectués lors du processus d'évolution. Ce journal d'évolution sera également réutilisé pour détecter les inconsistances générées du système.

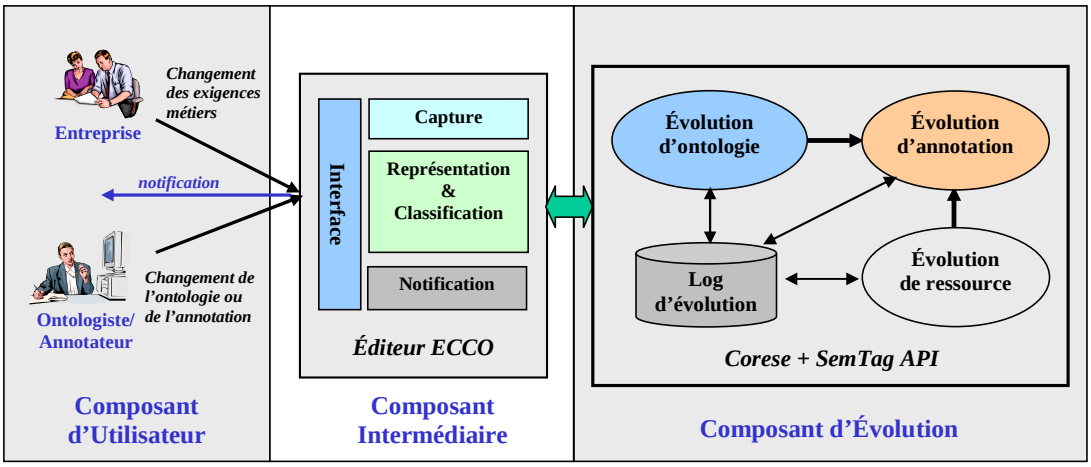


Figure 15 - Architecture du système CoSWEM

3.2.2 Description des composants du système

Composant d'Utilisateur

Ce composant gère les rôles possibles des agents humains qui interagissent avec le système CoSWEM. Plusieurs profils d'utilisateur existent et sont gérés par CoSWEM. Chaque profil permet d'afficher à l'utilisateur (jouant ce rôle) des vues qui correspondent aux types de tâches qu'il réalise dans son travail. Il est nécessaire de distinguer les rôles différents des profils d'utilisateur impliqués du système. Nous distinguons les rôles de différents types d'utilisateur ci-dessous :

- ***Utilisateur :*** C'est lui qui utilise le système de gestion des connaissances pour manipuler les connaissances de son domaine métier. Il aurait besoin des fonctions de gestion de l'évolution intégrées dans CoSWEM pour surveiller ou gérer les changements produits dans son système de gestion des connaissances.

- ***Ontologiste (Fournisseur d'ontologie) :*** Les ontologies utilisées dans le système de gestion de connaissances sont créées par les ontologistes (ou par les fournisseurs d'ontologie). Ce type d'utilisateur représente des personnes qui comprennent le domaine métier et modélisent ce domaine sous forme d'une représentation formelle et consensuelle. Au cours de l'évolution, ces ontologistes peuvent modifier certaines parties de l'ontologie pour s'adapter aux nouveaux besoins de l'entreprise.

- ***Annotateur (Ingénieur d'annotation) :*** Les annotateurs utilisent les ontologies ainsi créées pour annoter les documents, les processus ou les ressources de l'entreprise en reposant sur les concepts et relations définis dans ces ontologies.

- ***Ingénieur du système :*** C'est lui qui contrôle le système CoSWEM et effectue les changements nécessaires pour assurer la consistance de tout le système de gestion de connaissances. Un tel ingénieur est responsable de la détection et de la correction des inconsistances générées avec l'aide de l'outil CoSWEM. Dans notre point de vue, l'Ingénieur du système peut être considéré comme l'Administrateur qui gèle l'outil CoSWEM installé sur le serveur web Tomcat. Il paramétrise également l'outil et gère les groupes d'utilisateurs impliqués du système.

Composant Intermédiaire

- ***Module Interface :*** Ce module s'occupe des aspects d'interaction avec l'utilisateur. Il permet aux utilisateurs d'exprimer leurs besoins de visualisation et d'accéder aux ontologies et annotations sémantiques. Les utilisateurs peuvent surveiller la différence entre des versions de l'ontologie, les annotations inconsistantes... et aussi guider le processus de correction des inconsistances grâce aux interfaces de ce module.

- ***Module Représentation des changements :*** Les changements ontologiques effectués sont représentés d'une manière formelle dans ce module. Nous établissons une ontologie d'évolution qui décrit tous les types de changements ontologiques possibles. Nous les examinons de façon plus détaillée dans le chapitre 4.

- ***Module Notification :*** Ce module permet à l'utilisateur de savoir quels sont les changements au niveau de la configuration du système, par exemple la confirmation de changer les chemins des sources d'ontologie ou d'annotation sémantique utilisées dans le système, ou de recharger le moteur d'inférence du système, etc. D'autre part, après avoir effectué des changements, l'utilisateur sera également notifié de cette mise à jour.

Composant d'Evolution

Nous avons décrit les trois modules principaux (i.e. ressource, ontologie et annotation sémantique) du composant d'évolution dans l'architecture du système CoSWEM (c.f. Figure 15). Toutefois, dans le cadre du travail de thèse, nous nous intéressons à l'évolution de l'ontologie et de l'annotation sémantique seulement. Cependant, étant donné que nous voudrions esquisser une architecture globale et générale de système de gestion d'évolution des composants du Web sémantique, nous prendrons en compte le module d'évolution des ressources dans le chapitre sur les perspectives de notre travail.

- ***Module Évolution d'ontologie :*** Ce module nous permet de travailler avec différentes versions de l'ontologie. Dans ce module, nous ne traiterons pas la sémantique du changement de l'ontologie mais nous nous concentrerons sur la différence au niveau conceptuel entre des versions de l'ontologie et la propagation des changements de l'ontologie vers les annotations sémantiques concernées. Nous formalisons les changements effectués et capturés pour être réutilisés ultérieurement dans le module d'évolution de l'annotation.

- ***Module Évolution d'annotation :*** Après avoir reçu les informations sur les changements ontologiques propagés, ce module d'évolution de l'annotation détectera les annotations sémantiques devenues obsolètes par rapport à la nouvelle version de l'ontologie à cause des changements effectués. Ces annotations sémantiques obsolètes sont ensuite corrigées selon le choix de solution de l'utilisateur pour assurer la consistance de toute la base d'annotations.

- ***Journal d'évolution :*** Au cours de l'évolution, nous avons besoin d'un mécanisme qui sauvegarde la trace des changements effectués. Avec un journal d'évolution, nous pouvons garder l'historique des changements, des éléments affectés, l'ordre d'exécution de ces changements, etc. Ces informations, une fois sauvegardées, sont transformées en annotations sémantiques, cela nous permet de les manipuler facilement pour découvrir les informations utiles concernant le processus d'évolution.

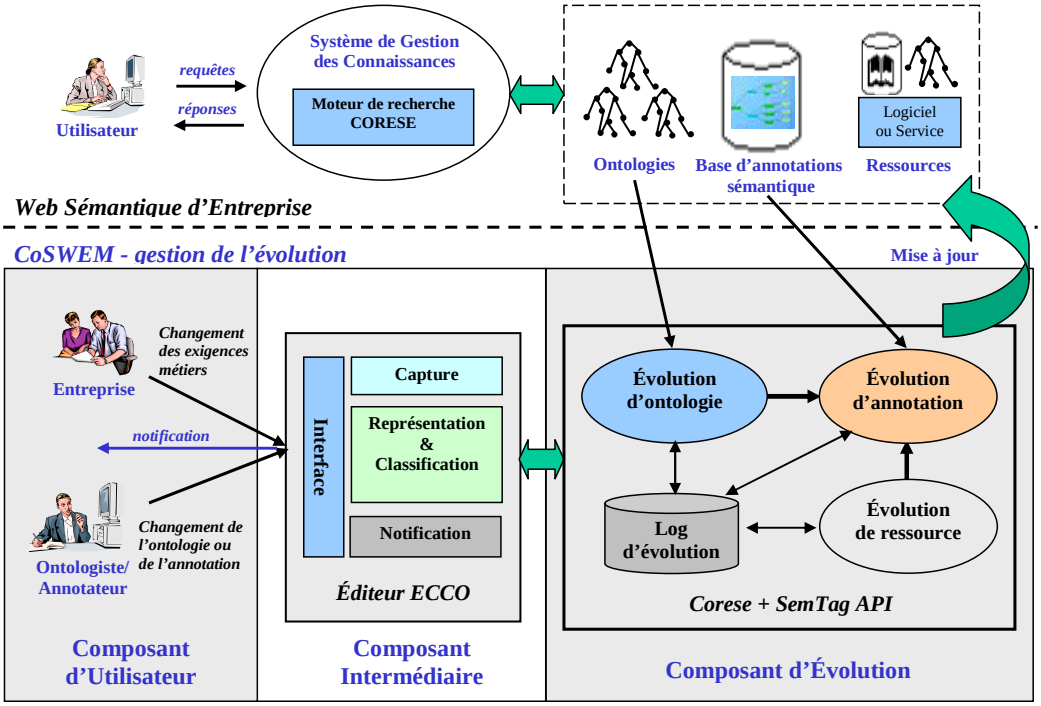


Figure 16 - Intégration de CoSWEM dans un Web sémantique d'entreprise

3.2.3 Intégration de CoSWEM dans un Web sémantique d'entreprise

Nous nous intéressons dans cette section à l'intégration du système de gestion de l'évolution CoSWEM dans un Web sémantique d'entreprise. La partie supérieure (c.f. Figure 16) représente un Web sémantique d'entreprise qui contient des sources de données évolutives. Ces parties évolutives seront ajustées dans la partie inférieure pour assurer la cohérence de tout le système (c.f. Figure 16).

Le composant d'évolution a besoin de charger les différentes versions des ontologies évolutives et la base d'annotations sémantiques du Web sémantique d'entreprise. Après avoir exécuté des changements et résolu des inconsistances générées dans les annotations, CoSWEM doit appliquer la phase de mise à jour en vue de renouveler toutes les annotations obsolètes par rapport à la nouvelle version de l'ontologie correspondante. Cette mise à jour sera aussi notifiée aux utilisateurs.

3.3 Évolution de l'ontologie dans le système CoSWEM

Dans le chapitre 2, nous avons fait un résumé de certaines recherches [Stojanovic, 2004], [Rogozan et al., 2005], [Plessers, 2006] qui ont étudié l'évolution de l'ontologie. Ces recherches ont mentionné deux phases

importantes : (i) sémantique du changement et (ii) propagation du changement de l'ontologie. Dans notre approche de l'évolution de l'ontologie, nous nous intéressons à cette dernière en soulignant la propagation des changements ontologiques vers les annotations concernées. D'autre part, nous nous focalisons également sur certains aspects qui assistent le processus de la propagation des changements.

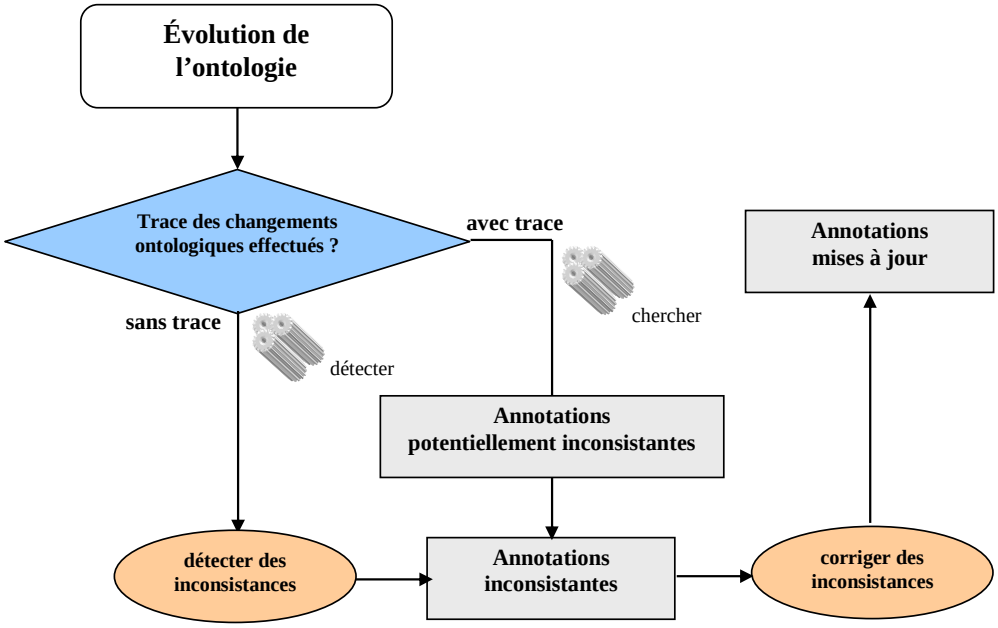


Figure 17 – Propagation des changements de l'ontologie dans le système CoSWEM

3.3.1 Propagation des changements de l'ontologie

Après l'application des changements sur une ontologie, cette ontologie a évolué vers une nouvelle version. Nous distinguons les deux cas d'évolution de l'ontologie qui peuvent influencer l'état de consistance de l'annotation : (i) avec trace et (ii) sans trace des changements réalisés entre deux versions de l'ontologie.

Propagation des changements avec la trace

Dans ce cas, nous pouvons garder la trace de changement entre les deux versions O_1 et O_2 de l'ontologie (c.f. Figure 17). Tous les changements réalisés ainsi que les résultats des opérations sont conservés dans un journal de changement *trace*($O_1 \rightarrow O_2$). Ces changements réalisés pourraient être représentés d'une manière plus formelle selon notre classification de changement (simple et composite). Parmi les annotations sémantiques affectées par ces changements, il pourrait exister les annotations consistantes et aussi inconsistantes. Nous les appelons comme les annotations potentiellement inconsistantes. Nous cherchons ensuite des annotations vraiment inconsistantes en vérifiant les types de changements conservés dans le journal. Nous pouvons passer sur le changement est de type "correction facultative" (i.e. il n'entraîne pas l'inconsistance sur

l'annotation). Si le changement est de type "correction obligatoire" c'est-à-dire qu'il entraîne l'inconsistance des annotations sémantiques, il sera traité dans la phase suivante. Enfin, nous appliquons des stratégies d'évolution correspondant à chaque changement ontologique afin de rétablir dans un état consistant les annotations sémantiques influencées. Tous ces termes seront clarifiés concrètement grâce à des définitions et des exemples dans le chapitre suivant (c.f. section 4.2).

Propagation des changements sans trace

Pour propager des changements de l'ontologie vers leurs annotations sémantiques dépendantes dans le cas où la trace des changements ontologiques n'est pas gardée, nous proposons une approche basée sur des règles permettant de vérifier l'état de consistance de l'annotation par rapport à la nouvelle version de l'ontologie. Ces règles portent sur différents éléments tels que le concept, la propriété, le domaine ou co-domaine, et le datatype.

3.3.2 Principaux aspects de l'évolution de l'ontologie

Représentation des changements ontologiques

La trace des changements réalisés entre deux versions de l'ontologie pourrait être représentée d'une manière plus formelle selon le type de changement (élémentaire et composite). Pour représenter ces changements, nous utilisons les termes définis dans l'ontologie d'évolution. La trace de ces changements contient non seulement des types de changements mais aussi des éléments (i.e. les concepts ou les propriétés) de l'ontologie du domaine qui sont considérés comme les paramètres de chaque changement.

Construction de l'ontologie d'évolution

En vue de représenter formellement des changements ontologiques, nous construisons une ontologie d'évolution qui repose sur une classification des changements possibles. Cette classification des changements peut reposer sur différents critères, par exemple sur la granularité de changement (i.e. élémentaire ou composite), sur le type de changement (i.e. changement de concept ou de propriété) ou sur l'influence du changement sur les annotations sémantiques (i.e. correction obligatoire ou facultative).

Dans cette ontologie, nous avons des propriétés décrivant les informations sur le processus d'évolution et de changement de l'ontologie telles que l'identificateur de version ontologique, l'auteur et la date à laquelle est effectuée la modification de l'ontologie, la trace de modification, etc. Toutes ces informations nous facilitent la tâche de description de la trace de changements en la rendant plus déclarative et complète.

Trace de changement

En reposant sur les concepts définis de l'ontologie d'évolution, nous établissons la trace décrivant les changements effectués sous forme d'une annotation. Les éléments de cette trace décrivant les types de changements sont des instances de l'ontologie d'évolution. Les paramètres du changement de la trace concernent les concepts ou les propriétés de l'ontologie de domaine (e.g. le changement `Renommer("ACTIVITY","Activity")` effectue une opération de renommage du concept `ACTIVITY` de l'ontologie de domaine) Cette trace sera crée automatiquement par ECCO [33] qui est un éditeur collaboratif et contextuel d'ontologie.

Stratégies d'évolution de l'ontologie

Bien que nous ne traitions pas actuellement la phase de résolution de changement ontologique dans CoSWEM, nous établissons par contre les stratégies d'évolution pour tous les types de changement de l'ontologie que nous avons proposés. Ces stratégies seront implémentées dans notre système pour servir au module de l'édition de l'ontologie.

Parallèlement à chaque stratégie d'évolution pour un changement de l'ontologie, nous avons établi la stratégie possible qui résoud la propagation du changement ontologique vers leurs annotations sémantiques afin de conserver l'état consistant pour ces annotations.

3.4 Évolution de l'annotation sémantique dans le système CoSWEM

Les modifications dans une partie de l'ontologie non seulement peuvent entraîner des inconsistances dans d'autres parties de la même ontologie, dans des ontologies dépendantes et dans les applications utilisant cette ontologie modifiée, mais aussi peuvent particulièrement affecter les annotations sémantiques qui utilisent les concepts ou les propriétés définis dans cette ontologie de base.

La tâche principale de l'évolution de l'annotation sémantique permet de détecter et de résoudre des inconsistances générées d'une manière systématique, assurant la consistance de toute la base d'annotations ainsi que les ontologies concernées. Les phases de détection et de résolution des annotations inconsistantes dépendent de l'évolution de l'ontologie, ou plus précisément elles dépendent de la disponibilité de la trace d'exécution des changements ontologiques. Nous présentons dans cette section nos approches principales pour réaliser l'évolution de l'annotation sémantique en général et dans le système CoSWEM en particulier.

[33] http://argentera.inria.fr/ecco-ewok

3.4.1 Étapes principales

Récupération des annotations potentiellement inconsistantes

Dans une base d'annotations, les annotations sémantiques peuvent reposer sur une ou plusieurs ontologies différentes. Les modifications faites sur l'ontologie n'affectent peut-être pas toute la base d'annotations mais seulement certaines annotations qui deviendraient inconsistantes. Nous distinguons dans notre approche les changements ontologiques de (i) correction obligatoire qui influencent la consistance des annotations sémantiques et de (ii) correction facultative qui pourraient ne pas affecter la consistance des annotations sémantiques. Une trace de changement de l'ontologie peut donc contenir à la fois ces deux types de changements entraînant les corrections obligatoires et facultatives.

Considérons les annotations potentiellement inconsistantes, celles-ci sont dues aux changements ontologiques effectués. Mais cela n'a pas forcément entrainé de vraies inconsistances : par exemple si nous effectuons des opérations d'ajout d'un nouveau concept à l'ontologie, alors l'état consistant de la base d'annotations est encore gardé. En vérifiant l'ensemble des annotations potentiellement inconsistantes, nous allons trouver des annotations vraiment inconsistantes.

Détection des annotations inconsistantes

Nous avons indiqué deux cas d'évolution de l'ontologie dans la section 3.3. Correspondant à chaque cas, nous proposons une méthode équivalente pour détecter les annotations inconsistantes. Avec la trace des changements, nous pouvons trouver les changements qui vont affecter la consistance de l'annotation et ensuite détecter les annotations inconsistantes à cause de ce changement en la comparant avec la liste des changements ontologiques de correction obligatoire. Dans le cas où on ne garde plus la trace de changement, nous appliquons une approche de détection d'inconsistance basée sur des règles pour trouver des annotations inconsistantes. Cette phase est réalisée d'une manière automatique sur les différents critères tels que l'inconsistance sur le concept, sur la propriété, sur le domaine ou co-domaine et sur le type de donnée.

Résolution des annotations inconsistantes

Après avoir détecté des annotations inconsistantes, celles-ci doivent être corrigées en vue de rétablir la cohérence de toute la base d'annotations. Pour chaque changement réalisé, nous proposons des stratégies d'évolution permettant de résoudre les inconsistances. Nous avons d'une part une option qui est de mettre en défaut l'exécution d'une stratégie d'évolution correspondant à chaque changement équivalent. D'autre part, les utilisateurs peuvent choisir l'opération à appliquer qui leur semble être la plus convenable à leurs besoins au cours de la

résolution des inconsistances. Nous pouvons réaliser cette phase d'une manière semi-automatique avec l'intervention de l'utilisateur.

3.4.2 Approches de détection et de résolution des inconsistances

Il serait impossible de gérer manuellement les effets des changements de l'ontologie vers leurs annotations sémantiques concernées car la taille et la complexité des ontologies et des annotations sémantiques peuvent parfois dépasser la capacité humaine. Pour la détection des annotations inconsistantes, nous avons besoin d'un mécanisme de détection automatique car si cela est laissé à la charge d'un ingénieur d'ontologie, le processus d'évolution sera inexact et long. En plus, il est peu réaliste de s'attendre à ce que les humains puissent comprendre toute l'ontologie, les annotations concernées ainsi que les inconsistances générées à cause des modifications. La phase de résolution sera réalisée automatiquement par un choix par défaut ou d'une manière semi-automatique avec l'intervention de l'utilisateur. Afin de répondre aux besoins de ces phases, nous proposons deux approches pour l'évolution de l'annotation sémantique :

- Approche procédurale : cette approche est appliquée dans le cas où il existe un journal de trace qui capture des changements ontologiques effectués.

- Approche basée sur des règles : cette approche est dédiée aux cas où l'on ne garde plus la trace de changement entre les versions de l'ontologie.

3.4.2.1 Approche procédurale : évolution avec le trace de changement

Dans le cas d'évolution de l'ontologie où on peut garder la trace de changement *trace(O_1 , O_2)* entre les deux versions ontologiques O_1 et O_2 (c.f. Figure 17), tous les changements réalisés, l'ordre d'exécution de ces changements ainsi que les résultats des opérations sont conservés dans un journal de changement (ou journal d'évolution).

Nous proposons de traiter l'évolution dans ce cas par une approche procédurale qui décompose suivant des étapes principales (c.f. Figure 18):

- ***Récupérer des annotations potentiellement inconsistantes :*** Nous utilisons dans cette étape un moteur de recherche sémantique Corese développé par l'équipe Acacia [Corby et al., 2004] pour faire des interrogations sur les bases d'annotations. Corese peut trouver les annotations potentiellement inconsistantes qui peuvent inclure des annotations consistantes ou inconsistantes concernées.

- ***Trouver les annotations inconsistantes :*** D'une manière générale, nous pouvons vérifier les contraintes de consistance de l'ontologie pour trouver des annotations vraies inconsistantes à partir d'un ensemble d'annotations potentiellement inconsistantes. Dans notre cas, nous pouvons trouver les annotations inconsistantes en vérifiant si elles contiennent des changements du type de correction obligatoire (c.-à-d. ceux qui détruisent l'état consistant d'une annotation sémantique)

- ***Appliquer les stratégies d'évolution :*** Nous avons construit des stratégies d'évolution possibles pour chaque changement ontologique ainsi que pour les annotations sémantiques correspondantes. Dans cette étape, nous pouvons appliquer des stratégies d'évolution correspondant à chaque changement ontologique ou laisser le choix à l'utilisateur d'une opération convenable afin de rétablir l'état consistant pour les annotations sémantiques influencées.

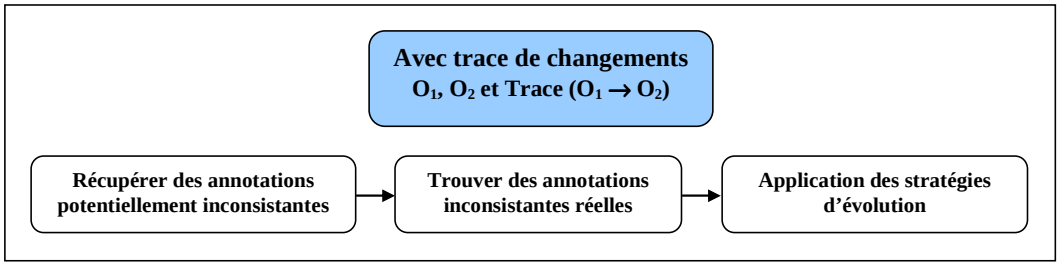


Figure 18 - Approche procédurale pour l'évolution avec le trace de changement

3.4.2.2 Approche basée sur des règles : évolution sans trace de changement

Le deuxième cas a souvent lieu dans le contexte dynamique et distribué du Web sémantique, car il n'est pas toujours possible de garder les traces entre des versions de l'ontologie (c.f. Figure 17).

Dans ce cas, nous pouvons réutiliser les résultats des recherches existantes (c.f. Figure 19) permettant de trouver les similarités et les différences diff(O_1,O_2) entre deux versions O_1 et O_2 de l'ontologie ainsi que les changements réalisés entre ces versions [Noy et Musen, 2002] [Klein, 2004]. Selon ces approches, nous pouvons obtenir une liste des différences des éléments de l'ontologie mais cette liste pourrait ne pas toujours être identique avec les changements effectués de l'ontologie. Autrement dit, cette liste de différences n'assure pas l'ordre des exécutions ainsi que les éléments affectés par les changements ontologiques. Par conséquent, il est indispensable d'avoir une étape intermédiaire qui nous permet de raffiner la liste de différences pour détecter quels sont les changements qui ont été effectués. Avec ces changements ontologiques détectés, nous pouvons suivre ensuite les procédures de résolution des inconsistances présentées comme nous avons indiqué dans le cas précédent (c.f. Figure 18).

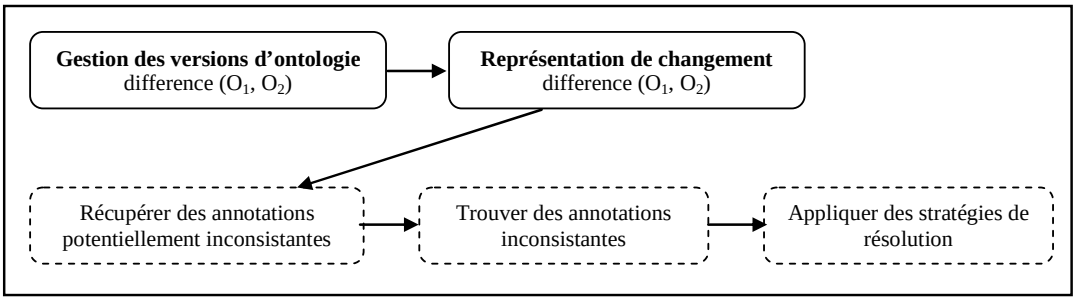


Figure 19 - Approche de gestion des versions d'ontologie

En effet, la méthode de réutilisation de ces recherches existantes ne nous fournit pas de meilleurs résultats car ces travaux effectuent principalement une comparaison des versions de l'ontologie au niveau syntaxique. Le résultat final ne serait pas exact si nous travaillons avec des versions de l'ontologie dont la taille est grande ou complexe. En plus, il faut passer ensuite par une étape d'extraction des changements ontologiques qui pourrait parfois générer des problèmes imprévus. Cela empêchera le processus de détection des annotations inconsistantes concernées de fonctionner.

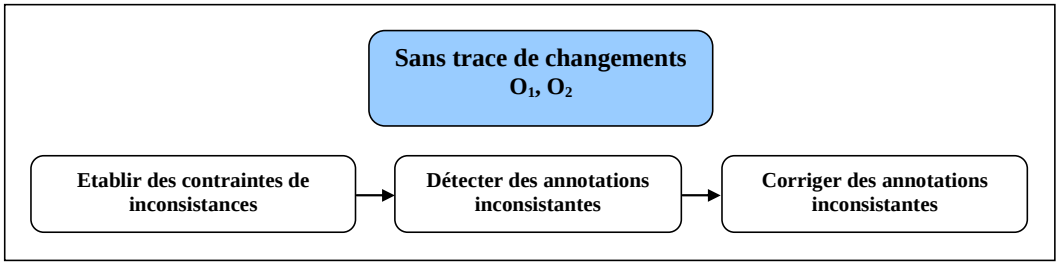


Figure 20 - Approche basée sur des règles pour l'évolution sans trace de changement

Pour pallier à cet inconvénient, nous proposons une approche basée sur des règles qui nous permet de détecter des annotations inconsistantes directement par rapport à la nouvelle version de l'ontologie (c.f. Figure 20). Cette approche repose sur les étapes principales suivantes :

- ***Établir les contraintes de consistance :*** qui assurent l'état cohérent d'une annotation sémantique par rapport à une ontologie de base. Une annotation devient inconsistante si l'une de ses contraintes de consistance est violée.

- ***Détecter des annotations inconsistantes :*** Nous établissons un ensemble de règles de détection sur les aspects du concept, de la propriété, du domaine ou co-domaine et du datatype. Toutes ces règles sont vérifiées si elles satisfont les contraintes de consistance définies pour trouver les types d'inconsistances différentes qui se passent dans la base d'annotations. Nous présentons d'une manière détaillée ces règles de détection dans le chapitre 5.

- ***Corriger des annotations inconsistantes :*** Après avoir détecté les annotations inconsistantes, elles seront corrigées en appliquant les règles de correction d'inconsistance. Les utilisateurs peuvent alors choisir, dans cette phase, l'opération à appliquer qui leur semble être la plus convenable à leurs besoins.

3.5 Outils de support utilisés dans le système CoSWEM

Dans cette section, nous présentons des outils de support qui sont utilisés dans le système CoSWEM.

3.5.1 CORESE - Moteur de recherche sémantique

Le moteur de recherche d'information est le noyau des applications d'extraction d'information. Dans le contexte du Web sémantique, les moteurs de recherche traditionnels basés sur des mots-clés ne rendent pas toujours de résultats pertinents et précis. Nos collègues de l'équipe Acacia ont développé un moteur de recherche sémantique Corese[34] (COnceptual REsource Search Engine) [Corby et al., 2004] qui est dédié à des applications de web sémantique organisationnel ou d'entreprise. Ce moteur de recherche nous permet de charger des ontologies représentées dans le langage RDFS et de faire des recherches d'information sur la base d'annotations qui utilise des termes définis dans ces ontologies.

Principes de Corese

Corese implémente un moteur RDF(S) basé sur les Graphes Conceptuels (GC) [Sowa, 1984], un formalisme de représentation qui offre des mécanismes de requêtes et d'inférences permettant de manipuler les connaissances. Grâce à la correspondance possible entre RDF et les Graphes Conceptuels, CORESE combine les avantages de ces deux formalismes de représentation des connaissances pour mieux exprimer et échanger des connaissances. Comme indiqué dans la Figure 21, les classes et les propriétés de RDFS sont traduites vers les types de concepts et les types de relations des GC (support) pour créer une base de RDF/GC. Chaque requête est formulée sous forme d'énoncés SPARQL, traduit en un graphe de requête qui est ensuite projeté sur la base des GC afin de trouver les graphes qui s'apparient avec lui. Corese intègre ainsi un interpréteur du langage de requêtes SPARQL dédié à RDF. Une fois la projection faite, les graphes résultats sont traduits en retour en RDF ou XML.

Architecture de Corese

[34] http://www-sop.inria.fr/acacia/soft/corese/

Ce moteur de recherche repose sur le langage RDF (Resource Description Framework), un des standards du web sémantique proposé par le W3C. Le langage RDF permet de décrire le contenu des documents à travers les annotations sémantiques qui sont basées sur une ontologie représentée en RDF Schema ou OWL-Lite (c.f. Figure 21). Dans l'architecture de Corese, grâce à un serveur web sémantique et à des interfaces dédiées à l'exploitation des informations, l'utilisateur peut interroger la base d'annotations (et recevoir par la suite des résultats de recherche) en utilisant le langage de requêtes SPARQL qui intègre plusieurs variables et opérateurs : opérateurs arithmétiques, négation et comparaison de types, etc. Pour inférer des résultats, Corese fait la projection d'une requête sur des graphes conceptuels afin d'extraire un graphe conceptuel résultat. Ce graphe résultat est traduit ensuite en un langage standard (RDF(S) ou XML) permettant ainsi sa présentation à l'utilisateur ou sa réutilisation par un autre programme. D'autre part, Corese possède aussi une fonction de recherche approchée qui permet de fournir une réponse proche dans le cas où aucune réponse exacte n'existe.

Enfin, en vue de compléter et d'enrichir la base d'annotations, Corese propose des règles de production permettant d'exploiter automatiquement des connaissances implicites à travers des inférences qui faciliteront le processus de recherche d'information.

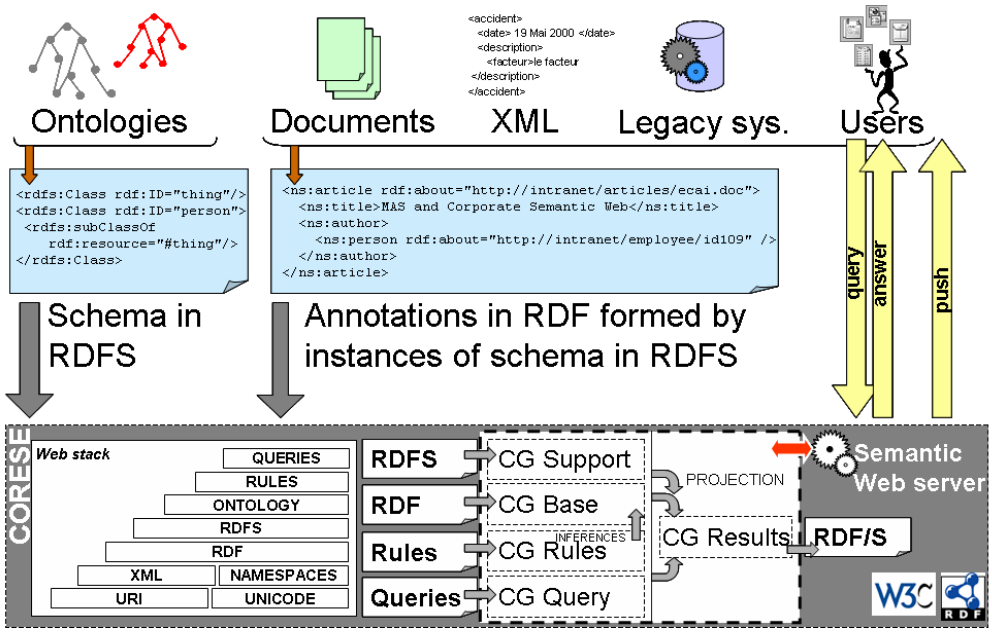


Figure 21 - Architecture de Corese

Applications utilisant CORESE

Le moteur de recherche sémantique Corese a été testé et intégré dans plusieurs applications réelles et outils tels que :

- CoMMA [Gandon, 2002]: un système multi-agents pour la gestion d'une mémoire organisationnelle. Corese permet de manipuler l'ontologie O'CoMMA contenant 472 concepts qui sont utilisés pour annoter des documents ou des ressources d'une organisation.

- Ligne-de-Vie [Dieng-Kuntz et al., 2004]: un staff virtuel pour un réseau de soins qui se base sur une ontologie contenant environ 26430 concepts. Il assiste les discussions entre les médecins sur des thérapies alternatives d'une pathologie donnée, selon les profils du patient.

- MEAT [Khelif, 2006]: une mémoire d'expériences (MEAT) permettant d'intégrer et de faire partager les connaissances du domaine des expériences de puces à ADN.

- OntoWatch [Cao, 2006] : un système de veille permet aux utilisateurs de faire des recherches sémantiques sur les annotations disponibles dans le système. Corese est aussi utilisé pour les services concernant l'ontologie, fournir l'accès à des concepts, des propriétés nécessaires pour les algorithmes de recherche et annotation des documents.

- SEWESE [Gandon et Durville, 2007] : un serveur Web sémantique permet la création dynamique d'applications Web à partir d'une base de connaissances représentées en RDF. Ce serveur sémantique est aussi disponible en bibliothèque de fonctions (API) utilisable pour les applications du web sémantique qui se basent sur des technologies standards du Web (XML, XSLT, JSP, etc.).

Dans notre système CoSWEM, qui permet de gérer l'évolution des composants du Web sémantique d'entreprise, Corese est aussi intégré pour manipuler les ontologies et les bases d'annotations disponibles dans la mémoire d'entreprise. Ce moteur de recherche nous permet d'interroger des requêtes sur la base d'annotations pour faciliter le processus de vérification des inconsistances générées dans les annotations sémantiques après l'évolution.

3.5.2 ECCO – Editeur Collaboratif et Contextuel d'Ontologie

L'éditeur d'ontologie ECCO[35] permet de créer, de façon collaborative et contextualisée, une ontologie pas à pas en suivant un ensemble d'étapes. Chaque étape apporte:

- un niveau de détail/précision supérieur à la précédente,

- un ensemble de fonctionnalités dédiées à une tâche précise (spécifique à l'étape en question).

[35] http://www-sop.inria.fr/edelweiss/projects/ewok/

ECCO suit le cycle de conception d'une ontologie à partir de termes choisis dans une source de données jusqu'à l'édition détaillée de l'ontologie finie. Le processus d'édition, au cours de ces différentes étapes, s'effectue collaborativement au sein d'une équipe partageant les sources de données et le vocabulaire qui en est extrait (c.f. Figure 22). Parallèlement à cela, il est possible d'ajouter des règles d'inférence qui viennent compléter l'ontologie, et d'effectuer des tests afin de vérifier que l'ontologie répond bien aux requêtes souhaitées.

Actuellement, l'éditeur ECCO est intégré et utilisé dans certains projets (par exemple E-WOK_HUB, Palette) afin de construire des ontologies au sein d'une équipe d'utilisateur collaboratif. Concernant notre travail et le système CoSWEM, l'éditeur ECCO nous fournit un mécanisme de gestion des modifications effectuées sur une ontologie. Après ces modifications, les changements ontologiques effectués sont capturés et générés sous forme d'une annotation RDF qui est basée sur une ontologie de gestion d'évolution développée au sein de l'équipe. Cette annotation générée sera ensuite utilisée pour détecter des annotations inconsistantes à cause des changements de l'ontologie dans le but de mettre à jour une base d'annotations relatives à l'ontologie modifiée.

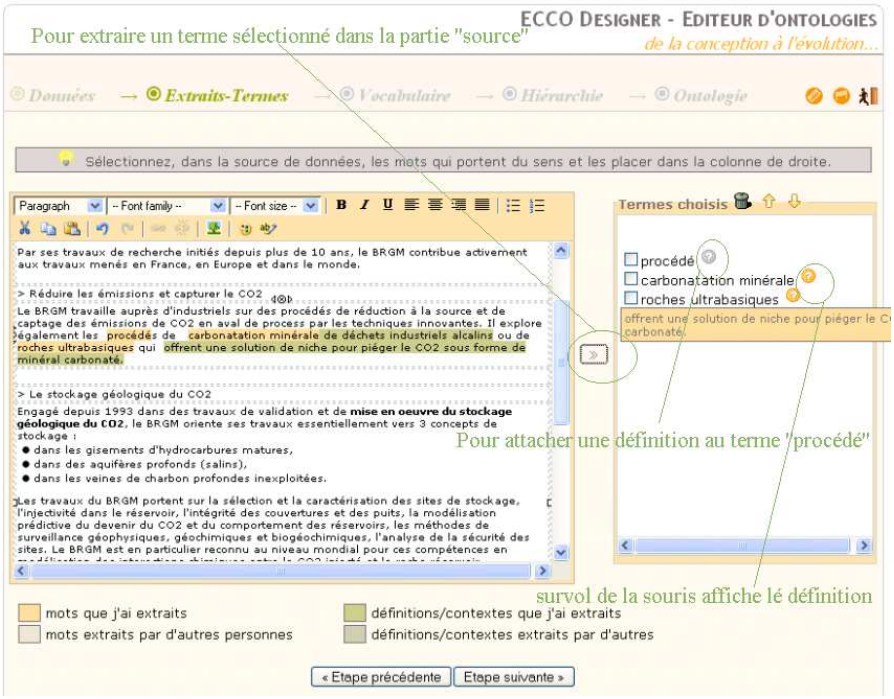


Figure 22 - Interface de l'éditeur d'ontologie ECCO

3.5.3 SemTag - Librairie de tags JSP sémantiques

Puisque le système CoSWEM est une application basée sur le Web, nous nous intéressons à des bibliothèques de fonctions (API) utilisables pour les applications du web sémantique qui se basent sur des technologies standards du Web tels que XML, XSLT, JSP/Servlet, etc.

Au sein de l'équipe, nos collègues ont développé SemTag[36], une librairie de tags JSP, qui permet d'utiliser les notions du web sémantique avec le moteur de recherche Corese dans les applications du web. Pour les aspects JSP, cette librairie s'appuie sur la librairie standard (JSTL) et elle est développée en Java 5 et JSP2.0. La librairie SemTag est combinée dans une plate-forme SeWeSe (c.f. Figure 23) permettant d'intégrer des opérations sur le Web sémantique (e.g. soumettre une requête SPARQL, transformer des résultats de requêtes...) en technologies du Web classique (e.g. des pages JSP, servlets...) [Gandon et Durville, 2007]. Le but d'une telle plate-forme est de fournir les primitives et les composants réutilisables, configurables et extensibles afin de réduire la quantité de temps dépensée pour développer de nouvelles applications du Web sémantique.

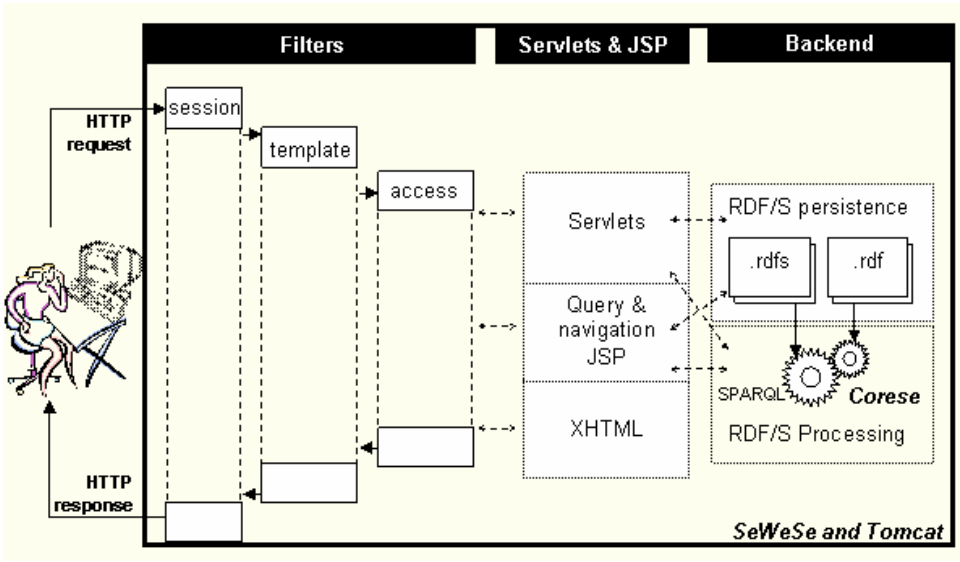


Figure 23 - Architecture de SeWeSe

Dans la SemTag, les tags JSP sont des extensions de la librairie standard JSP et ils ont pour but de réaliser des tâches différentes dans une application du Web sémantique. SemTag fournit certains tags tels que :

- Tags dédiés pour afficher et naviguer dans tout ou une partie du schéma RDFS.
- Tags dédiés pour créer des annotations sémantiques dans la base d'annotations.
- Tags dédiés pour modifier des annotations sémantiques existantes.
- Tags dédiés pour manipuler des triplets dans l'annotation.
- Tags dédiés pour faire des requêtes SPARQL.

[36] http://www-sop.inria.fr/acacia/soft/sewese/

- Tags dédiés pour les tâches d'administration (i.e. redémarrer le moteur), etc.

Nous avons utilisé certains tags dans le système CoSWEM pour quelques tâches spécifiques. Par exemple nous utilisons le tag `<stl:for-each-child>` afin d'afficher tous les sous-concepts (ou sous–propriétés) d'un concept (ou d'une propriété) donné(e) `root="${uri}"` dans une hiérarchie de l'ontologie (c.f. Figure 24).

```
<ul>
  <stl:for-each-child root="${uri}">
  <li>
    <a href="simple_query.jsp?uri=${stl:encode(current)}">
            ${current}
    </a>
  </li>
  </stl:for-each-child>
 </ul>
```

Figure 24 - Exemple d'un tag de la librairie SemTag

3.6 Conclusion

Nous avons donné dans ce chapitre une vision globale des principaux composants du système CoSWEM. A travers les besoins analysés, nous avons proposé ce système qui permet de gérer l'évolution du Web sémantique d'entreprise. Nous avons aussi présenté les outils de support utilisés dans CoSWEM pour effectuer certaines tâches spécifiques. Ce système implique plusieurs types d'utilisateur tels que l'utilisateur du système de gestion des connaissances, l'ingénieur ou le fournisseur d'ontologie, l'annotateur et l'ingénieur qui gère le système d'évolution.

Dans l'architecture du système CoSWEM, nous ne nous sommes pas concentrés sur l'évolution des ressources mais avons insisté particulièrement sur les deux composants principaux (i) l'évolution de l'ontologie et (ii) l'évolution de l'annotation sémantique. Nous présentons d'une manière détaillée ces deux composants ainsi que leurs fonctionnalités dans les deux chapitres suivants.

Chapitre 4

Évolution de l’ontologie

Les ontologies doivent normalement changer pour pouvoir s’adapter à l’évolution des conditions métiers. Plusieurs problèmes apparaissent quand on essaie d'utiliser ensemble des ontologies développées indépendamment, ou bien quand des ontologies existantes doivent être modifiées pour s’adapter à de nouveaux objectifs. La gestion de l’ontologie prend donc un rôle crucial. Elle nécessite un ensemble de méthodes et techniques permettant d’utiliser efficacement de multiples ontologies constituées à partir de sources différentes, de modifier des ontologies selon les nouvelles exigences ou de maintenir des variantes différentes de l’ontologie, etc. L’évolution de l’ontologie est considérée comme une partie de la gestion de l’ontologie facilitant la modification d'une ontologie en préservant sa consistance. Dans ce chapitre, nous allons étudier les aspects importants de l’évolution de l’ontologie à travers les trois étapes principales du processus d’évolution : (i) représentation des changements, (ii) résolution des changements et (iii) propagation des changements de l’ontologie. Par conséquent, ce travail est en relation étroite avec les niveaux d’inconsistance de structure. Nous allons nous focaliser sur le niveau structurel dans notre processus de l’évolution de l’ontologie Nous présentons les travaux réalisés pour chaque étape et leur intégration dans le système de gestion de l’évolution CoSWEM.

4.1 Démarche générale

Dans le domaine de l'intelligence artificielle, les ontologies sont développées pour faciliter le partage et la réutilisation de la connaissance. L'aide à l'évolution de l'ontologie est nécessaire dans presque toutes les situations où l'ontologie est utilisée dans des applications réelles [Davies et al., 2002]. Dans ces cas, des ontologies sont souvent développées par plusieurs personnes et continueront à évoluer à travers le temps, en raison des changements du monde réel, des adaptations à différentes tâches, ou des alignements avec d'autres ontologies. Pour que de tels changements n'empêchent pas l'utilisation existante de l'ontologie, nous avons besoin de gérer l'évolution de l'ontologie et les changements ontologiques effectués.

Se basant sur les recherches existantes sur l'évolution des schémas des bases de données, [Stojanovic, 2004] définit *l'évolution de l'ontologie comme une adaptation opportune d'une ontologie aux changements se produisant et à une propagation consistante de ces changements à leurs parties dépendantes.*

Il y a certaines raisons en réalité qui entraînent l'évolution de l'ontologie : la modification d'une ontologie initiale consistante peut provoquer des inconsistances dans les autres parties de l'ontologie, la réutilisation d'une ontologie qui est encore inconsistante, le remplacement d'une version ontologique par une autre version inconsistante, etc. Le contexte le plus typique est la modification d'une ontologie selon les requêtes de changements de l'utilisateur. Cela concerne étroitement notre scénario d'évolution dans lequel l'ontologie fournie est changée vers une autre version par l'ontologiste, nous devons les examiner et faire des modifications nécessaires pour assurer la consistance de l'utilisation des autres parties concernées par rapport à la nouvelle version de l'ontologie.

Pour la résolution des inconsistances, nous traiterons l'affectation des changements ontologiques sur les annotations sémantiques qui est en relation étroite avec les inconsistances de structure. Par exemple, un changement sur le concept (e.g. supprimer un concept, enlever un lien hiérarchique entre deux concepts…) ou sur la propriété (e.g. déplacer une propriété, renommer une propriété…) peut influencer l'état cohérent de l'annotation sémantique. Par conséquent, nous allons approfondir les inconsistances au niveau structurel dans notre travail de l'évolution de l'ontologie.

Nous présentons dans cette section le modèle de l'ontologie utilisé dans notre travail ainsi que les étapes de son processus d'évolution. Ces étapes seront détaillées dans les sections suivantes.

4.1.1 Modèle de l'ontologie

Il existe plusieurs définitions de l'ontologie et de ses modèles de représentation. Les définitions des modèles sont normalement divisées en deux catégories : générique et logique.

Dans notre travail, nous nous intéressons au modèle de représentation de l'ontologie générique. Nous visons en effet à proposer des modèles et des approches génériques pour résoudre des problèmes d'évolution, ces approches pourraient être exploitées ou adaptées selon les nouveaux besoins. Par conséquent, nous définissons le modèle de l'ontologie comme suit:

Définition 1- Modèle de l'ontologie

Une ontologie est le tuple $\mathbf{MO := (C, P, A, I, H_C, H_P, domain, range, name)}$

- $\mathbf{C}$ *: ensemble des concepts*
- $\mathbf{P}$ *: ensemble des propriétés*
- $\mathbf{A}$ *: ensemble des attributs*
- $\mathbf{I}$ *: ensemble des instances de concepts*
- $\mathbf{H_C \subseteq CxC}$ *: hiérarchie de concepts (relation acyclique)*
  Si $(c_1, c_2) \in \mathbf{H_C}$ *alors* c_1 *est un sous-concept de* c_2 *et* c_2 *est un super-concept de* c_1
- $\mathbf{H_P \subseteq PxP}$ *: hiérarchie de propriétés (relation acyclique)*
  Si $(p_1, p_2) \in \mathbf{H_P}$ *alors* p_1 *est une sous-propriété de* p_2 *et* p_2 *est une super-propriété de* p_1
- $\mathbf{domain}$ *: fonction qui rend l'ensemble des concepts constituant le domaine d'une propriété*
- $\mathbf{range}$ *: fonction qui rend l'ensemble des concepts constituant le co-domaine d'une propriété*
- $\mathbf{name}$ *: fonction qui rend le nom d'une entité de l'ontologie (e.g. nom du concept, nom de la propriété, etc.)*

Nous avons choisi le langage RDF(S) recommandé par le W3C [Miller et Manola, 2004] [Brickley et Guha, 2004] pour représenter notre ontologie.

Figure 25 - Processus d'évolution de l'ontologie

4.1.2 Processus d'évolution de l'ontologie

Nous nous intéressons à trois étapes principales dans le processus de l'évolution de l'ontologie (c.f. Figure 25)

- ***Représentation des changements :*** Afin de pouvoir résoudre des changements de l'ontologie, ces derniers doivent être identifiés et *représentés dans un format approprié*. Les changements ontologiques peuvent être représentés à différents niveaux de granularité.

- ***Résolution des changements :*** La résolution des changements est une phase dans le processus d'évolution de l'ontologie qui permet de *résoudre des changements* de l'ontologie d'une manière systématique en assurant la *consistance de toute l'ontologie*.

- ***Propagation des changements :*** La phase de propagation des changements doit *assurer la consistance des parties dépendantes après avoir exécuté une mise à jour de l'ontologie*. Ces parties dépendantes peuvent inclure des ontologies dépendantes, des instances, des annotations sémantiques concernées ainsi que des applications utilisant l'ontologie de base.

4.2 Représentation des changements de l'ontologie

4.2.1 Classification des changements ontologiques

Un changement de l'ontologie est considéré comme une action sur l'ontologie entraînant l'obtention d'une ontologie différente de la version originale. Nous avons présenté les types de changements et les classifications de plusieurs travaux différents dans la section 2.3.2. Dans cette partie, nous allons nous focaliser sur notre classification des changements ontologiques.

Classification des changements selon le niveau d'abstraction

Les changements peuvent être représentés à différents niveaux de granularité. [Klein, 2004] fait une distinction des changements ontologiques entre les différents niveaux d'interprétation d'une ontologie : (i) changement conceptuel, (ii) changement de spécification et (iii) changement de représentation. Une autre classification usuelle des changements de l'ontologie qui se base sur le niveau d'abstraction telle que les changements élémentaires, changements composites et changements complexes [Stojanovic, 2004].

Dans notre travail, nous nous intéressons aux types de changements *élémentaires* et *composites* car ils peuvent répondre à notre besoin d'exprimer les requêtes de changements sur l'ontologie et sur l'annotation sémantique. Il existe différentes définitions des changements élémentaires et composites, nous en donnons un aperçu ci-dessous.

Définition 2 – Changement élémentaire

Un changement élémentaire est un changement ontologique qui affecte une seule entité de l'ontologie.

Le changement élémentaire est parfois appelé changement simple; il est la base des changements implémentés dans la plupart des outils d'édition actuels de l'ontologie. Un exemple de changement élémentaire est présenté dans la section précédente (c.f. Figure 12, section 2.2.2), nous avons fait une opération de renommage du concept `ACTIVITY` vers `Activity`. L'ensemble des changements élémentaires proposés par [Stojanovic, 2004] [Klein, 2004] ne contient que des opérations *additives* et *soustractives* des concepts et des propriétés qui ne sont pas suffisants pour exprimer les relations entre les hiérarchies des concepts ou des propriétés, par exemple le changement `CreateHierarchyConceptLink (`c_1`,` c_2`)` décrit une opération de création d'un lien entre deux concepts c_1 et c_2, ou un autre changement `RemovePropertyDomainLink (c, p)` enlève le lien entre un concept `c` et une propriété `p` dans l'ontologie. Par conséquent, nous établissons une classification des changements élémentaires permettant de représenter tous les types de changements ontologiques possibles sur le concept et sur la propriété (c.f. Tableau 3).

Tableau 3 - Liste des changements élémentaires

Changements élémentaires	
1. sur le concept	**2. sur la propriété**
1.1 InsertConcept 1.2 DeleteConcept 1.3 RenameConcept 1.4 CreateHierarchyConceptLink 1.5 RemoveHierarchyConceptLink	2.1 InsertProperty 2.2 DeleteProperty 2.3 RenameProperty 2.4 CreateHierarchyPropertyLink 2.5 RemoveHierarchyPropertyLink 2.6 CreatePropertyDomainLink 2.7 RemovePropertyDomainLink 2.8 CreatePropertyRangeLink 2.9 RemovePropertyRangeLink

Cependant, cette granularité élémentaire du changement de l'ontologie n'est pas toujours appropriée pour les requêtes de changement plus compliqués ou les changements qui demandent une haute expressivité. Par exemple, si nous voulons diviser le concept `Governance_role` de l'ontologie O'CoP (c.f. Figure 12, section 2.2.2) en deux nouveaux concepts `Group_role` et `Community_role`, nous devons effectuer une série de changements élémentaires : créer deux nouveaux concepts `Group_role` et `Community_role`, attacher ces concepts au super-concept `Role_in_the_CoP` qui est aussi le concept père du `Governance_role`, transférer les propriétés et les instances concernées par le concept `Governance_role` sur les nouveaux concepts divisés, supprimer le concept `Governance_role`... Ce processus est complexe car nous devons effectuer une série d'opérations différentes qui pourraient augmenter la probabilité

de fautes au cours de la réalisation. L'intention des changements aurait parfois besoin d'être exprimée à un niveau plus élevé.

Par conséquent, nous utilisons également les changements composites qui peuvent représenter un groupe de changements élémentaires de l'ontologie appliqués ensemble. Considérant qu'un changement élémentaire peut être vu comme une modification isolée d'une ontologie, un changement composite définit un "contexte" d'évolution. Nous reprenons la notion de "voisinage" d'une entité de l'ontologie présentée dans [Stojanovic, 2004] qui est définie comme l'ensemble des concepts, des propriétés et des instances étant en relation avec cette entité. Pour illustrer cette notion de voisinage, nous revenons à l'extrait de l'ontologie O'CoP (c.f. Figure 12, section2.2.2), les entités voisines du concept `Member_role` sont les concepts `Role_in_the_CoP`, `Facilitator`, `Coordinator` et la propriété *`coordinate`* (c.f. Figure 26).

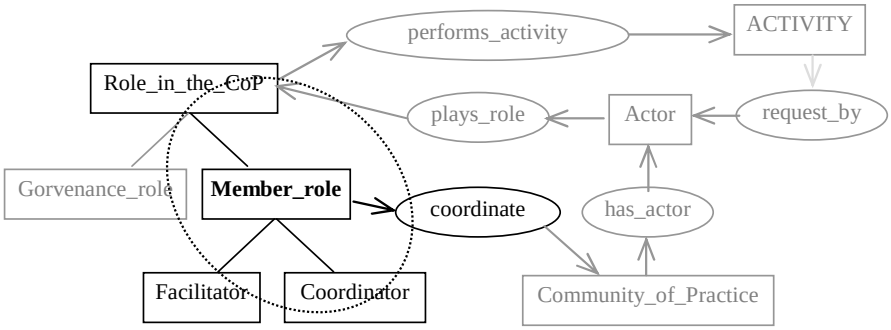


Figure 26 - Les voisins du concept `Member_role`

Définition 3 – Changement composite

Un changement composite est un changement ontologique qui affecte certaines entités de l'ontologie.

Selon cette définition, nous pouvons spécifier une requête de changement d'une manière plus flexible grâce à la possibilité d'expression plus significative du changement composite en comparant avec les changements élémentaires. On pourrait appliquer un seul changement composite au lieu d'effectuer plusieurs changements élémentaires pour obtenir le même but. En plus, les changements composites facilitent la phase de spécification des requêtes de changements car l'ontologiste n'a pas besoin de passer par chaque étape du changement élémentaire pour atteindre le même effet désiré. Cela permet au processus d'évolution de l'ontologie d'être réalisé plus rapidement, facilement et plus efficacement. En réalité, les changements composites sont définis et créés selon les besoins de l'utilisateur. Autrement dit, il n'est pas possible de définir un ensemble fixe de changements composites pour tout le monde. On peut toujours créer un nouveau changement composite en combinant des changements ontologiques différents ou en modifiant les paramètres du changement composite. Par exemple, pour la requête qui divise un concept, nous pouvons créer un changement composite pour le diviser en deux, en trois ou en plus de nouveaux

concepts selon notre besoin. Dans notre cas, nous établissons un ensemble des changements composites sur le concept et la propriété (c.f. Tableau 4). Ils peuvent couvrir des cas et des contextes de changement de l'ontologie qui peuvent affecter certaines entités de l'ontologie et entraîner les changements continus des annotations sémantiques concernées.

Tableau 4 - Liste des changements composites

Changements composites	
3. sur le concept	**4. sur la propriété**
3.1 CreateCommonConcept 3.2 MergeConcept 3.3 SplitConcept 3.4 InsertConceptGeneralisation 3.5 InsertConceptSpecialisation 3.6 MoveConcept	4.1 CreateCommonProperty 4.2 MergeProperty 4.3 SplitProperty 4.4 InsertPropertyGeneralisation 4.5 InsertPropertySpecialisation 4.6 MoveProperty

Pour chaque type de changement, nous décrivons d'une manière formelle la syntaxe, les paramètres et la sémantique de ce changement. Nous décrivons aussi les conditions avant et après qui doivent être satisfaisantes pour effectuer chaque changement. Basé sur le modèle de l'ontologie introduit dans la partie précédente, nous présentons (c.f. Tableau 5) un exemple d'une description du changement `CreateHierarchyConceptLink` (c_1, c_2) qui permet de relier deux concepts existants (appartenant à l'ensemble des concepts $\mathbf{C}$ de l'ontologie) pour créer un lien hiérarchique entre ces concepts $(c_1, c_2) \in \mathbf{H_c}$. La description complète des autres changements est présentée dans la partie Annexe A.

Tableau 5- Description d'un changement

Syntaxe	***Sémantique***
CreateHierarchyConceptLink (c_1, c_2)	Créer un lien de hiérarchie entre les concepts c_1 et c_2, c_2 devient le père de c_1
Pré-condition	***Post-condition***
$c_1, c_2 \in \mathbf{C}, (c_1, c_2) \notin \mathbf{H_c}$	$c_1, c_2 \in \mathbf{C}, (c_1, c_2) \in \mathbf{H_c}$

Classification des changements selon leur affectation aux annotations sémantiques

Puisque nous sommes intéressés par l'affectation des changements ontologiques vers les annotations sémantiques concernées, nous avons besoin de déterminer quels types de changements pourraient entraîner les inconsistances dans les instances ontologiques et les annotations sémantiques. Pour ce faire, nous divisons l'ensemble des changements de l'ontologie en deux groupes principaux (c.f. Tableau 6) : (i) *correction obligatoire* et (ii) *correction facultative*.

Définition 4 – Correction obligatoire

Une correction obligatoire entraîne nécessairement une ou des inconsistances dans les annotations concernées par ce changement.

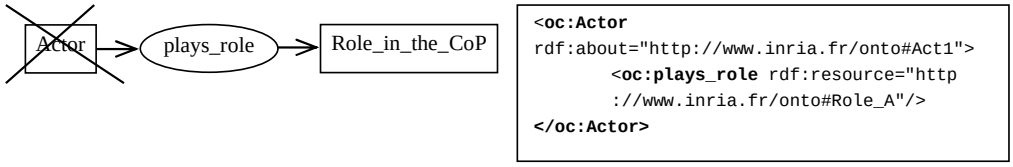


Figure 27 - Exemple d'un changement entraînant la correction obligatoire

La Figure 27 représente un type de changement entraînant la correction obligatoire. Supposons qu'il existe un triplet de l'annotation sémantique qui utilise les concepts `Actor`, `Role_in_the_CoP` et la propriété `plays_role` de l'ontologie de base. Le changement de suppression du concept `Actor` va causer une inconsistance sur l'annotation parce qu'elle contient un triplet dont le sujet est du type `Actor`.

Définition 5 – Correction facultative

Une correction facultative qui n'entraîne pas d'inconsistances dans les annotations concernées par ce changement.

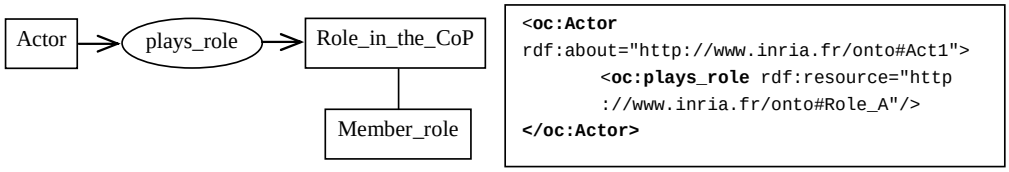


Figure 28 - Exemple d'un changement entraînant la correction facultative

Par exemple, nous faisons un ajout du concept `Member_role` à l'ontologie comme fils du concept `Role_in_the_CoP` (c.f. Figure 28). L'ajout de ce concept `Member_role` ne change rien au triplet de l'annotation sémantique reposant sur l'ontologie de base. Alors, ce type de changement d'insertion du concept est considéré comme un changement entraînant la correction facultative.

Nous faisons aussi un résumé des changements de l'ontologie (c.f. Tableau 6) selon les types de changements (i.e. correction obligatoire ou correction facultative), selon leur affectation (i.e. sur le concept, sur la propriété), ou selon le niveau d'abstraction (i.e. élémentaire ou composite).

Tableau 6 - Classification des changements entraînant la correction obligatoire et facultative

Changement	**Sur le concept**	**Sur la propriété**

Correction obligatoire	DeleteConcept RenameConcept RemoveHierarchyConceptLink	DeleteProperty RenameProperty RemoveHierarchyPropertyLink RemovePropertyDomainLink RemovePropertyRangeLink
	MergeConcept SplitConcept MoveConcept	MergeProperty SplitProperty MoveProperty
Correction facultative	InsertConcept CreateHierarchyConceptLink	InsertProperty CreateHierarchyPropertyLink CreatePropertyDomainLink CreatePropertyRangeLink
	CreateCommonConcept InsertConceptGeneralisation InsertConceptSpecialisation	CreateCommonProperty InsertPropertyGeneralisation InsertPropertySpecialisation

4.2.2 Construction de l'ontologie d'évolution

Afin de représenter ces changements d'une manière formelle, nous proposons de construire une *ontologie d'évolution* qui définit formellement la classification des changements ontologiques ainsi que leurs relations entre ces changements avec les entités de l'ontologie du domaine. Cette ontologie formalise également des informations concernant le processus d'évolution de l'ontologie. Dans la partie suivante, nous allons présenter les méthodes utilisées pour établir cette ontologie d'évolution.

Méthodes utilisées pour la construction de l'ontologie d'évolution

Depuis quelques années, la construction de l'ontologie est un sujet de recherche important dans le domaine de l'intelligence artificielle. Il existe plusieurs méthodologies et méthodes différentes nous permettant de faciliter la construction d'une ontologie. Nous pouvons classifier ces recherches existantes selon les différents critères [Gómez-Pérez et al., 2004] tels que (i) la construction à partir de zéro ou la réutilisation d'une autre source (ii) la manière de construire (automatique, semi-automatique, manuelle) ou (iii) le cycle de vie de l'ontologie, etc. Le choix d'une méthodologie ou d'une méthode de construction de l'ontologie dépend des besoins de l'ontologiste, de la taille de l'ontologie à développer ou de la capacité d'expression de l'ontologie… mais la plupart de ces méthodes possèdent les étapes principales suivantes:

- Analyser et identifier les objectifs de l'ontologie.
- Construire l'ontologie (capturer, représenter et intégrer des connaissances).
- Evaluer, tester et valider l'ontologie.

Dans notre cadre de travail, nous avons besoin de formaliser toutes les informations et les connaissances concernées par l'évolution de l'ontologie et de l'annotation sémantique. Puisque la taille de l'ontologie d'évolution n'est pas très grande et que les contraintes d'utilisation de cette ontologie sont minimes, nous décidons d'appliquer l'approche de construction de l'ontologie proposée par [Uschold et Gruninger, 1996]. Cette approche est similaire aux méthodologies de construction présentées par [Fernandez et al., 1997] et [Gómez-Pérez et al., 2004] qui couvrent également les étapes principales mentionnées ci-dessus.

Analyse et identification des parties principales

Le rôle principal de l'ontologie d'évolution est de définir les types de changements possibles et les informations relatives qui pourraient être produites au cours de l'évolution de l'ontologie. En utilisant cette ontologie d'évolution, un ontologiste peut spécifier une requête de changement d'une manière claire, expressive et sans ambiguïté. En vue de répondre à ces exigences, d'après nous, cette ontologie doit assurer les points suivants:

- Bien distinguer la classification des changements élémentaires, des changements composites ainsi que des changements entraînant la correction obligatoire et facultative. Ces changements vont décrire d'une manière formelle des actions qui ont été effectuées dans l'ontologie de domaine.

- Préciser les paramètres nécessaires pour chaque changement de l'ontologie. Par exemple les paramètres du changement `RenameConcept(`c_1`,` c_2`)` sont c_1 et c_2 qui correspondent respectivement au nom de l'ancien concept et à celui du nouveau.

- Pouvoir annoter les informations du processus d'évolution sous forme d'annotations sémantiques.

- Fournir des informations additionnelles qui éclairent le processus d'évolution ainsi que les changements exécutés, par exemple la version de l'ontologie modifiée, la date de modification, l'auteur des changements, etc.

En vue de répondre aux besoins spécifiques ci-dessus, les parties principales de l'ontologie d'évolution doivent comporter:

- une partie dédiée aux classifications des changements ontologiques,

- une partie dédiée aux tâches de spécification des paramètres de chaque changement,

- une partie dédiée à la trace des changements ontologiques exécutés,

- des propriétés supplémentaires et leurs valeurs associées pour représenter les informations additionnelles de l'évolution.

Construction et utilisation de l'ontologie d'évolution CoSWEM

Notre processus de construction de l'ontologie d'évolution CoSWEM comprend les étapes principales suivantes:

- Analyser les besoins de conceptualisation et de spécification du processus d'évolution ainsi que les étapes principales. Après cette phase, nous détectons le besoin d'avoir un mécanisme de représentation des connaissances décrivant les informations de changements. Nous établissons une ontologie d'évolution qui modélise les changements possibles de l'ontologie.

- Déterminer des concepts et des propriétés de l'ontologie d'évolution. En nous basant sur la classification des types de changements, nous avons déterminé les principaux concepts et propriétés de l'ontologie d'évolution.

- Représenter les concepts et les propriétés déterminés dans le langage RDF(S). Nous avons choisi le langage RDF(S) [Miller et Manola, 2004] recommandé par le W3C pour sa capacité d'expression des connaissances d'une manière formelle et claire. Ce langage est largement utilisé dans plusieurs applications et outils dédiés dans le domaine du Web sémantique. Nous l'utilisons aussi pour représenter des annotations sémantiques dans notre travail.

- Tester et utiliser l'ontologie d'évolution. Nous utilisons cette ontologie d'évolution CoSWEM dans certains projets pour lesquels la gestion de l'évolution est requise (par exemple le projet Palette[37]) afin de modéliser les connaissances concernant le processus d'évolution.

- Raffiner et améliorer l'ontologie d'évolution après les retours d'expériences d'utilisation. Après avoir créé et testé cette ontologie d'évolution, nous évaluons également son utilisation afin de raffiner et d'améliorer les parties de l'ontologie pour qu'elle puisse mieux modéliser d'autres variantes de scénarios et de contextes d'évolution.

[37] http://palette.ercim.org/

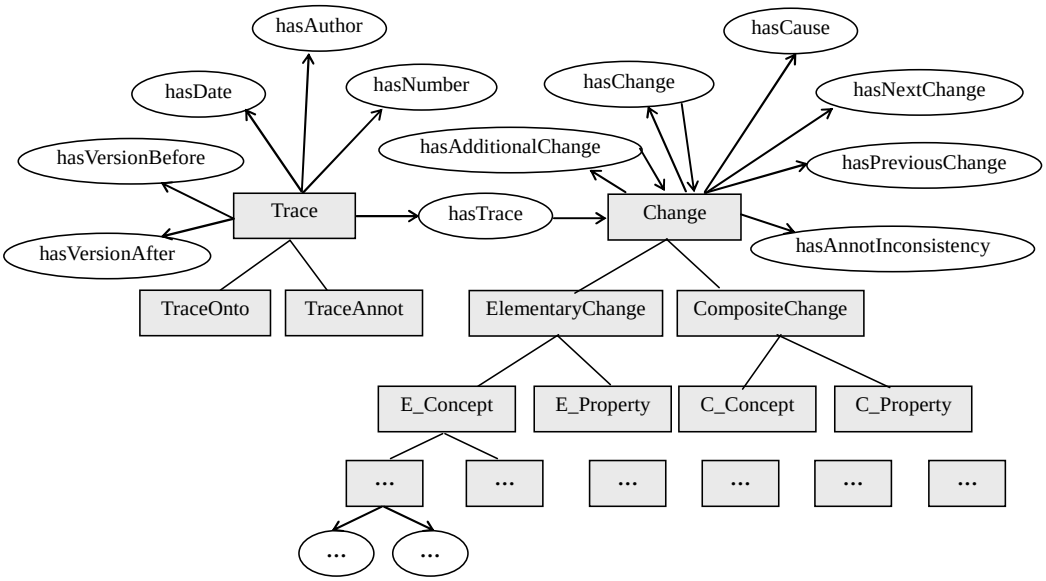


Figure 29 - Extrait de l'ontologie d'évolution CoSWEM

```
0   <?xml version="1.0" encoding="UTF-8"?>
1   <rdf:RDF
2   xmlns:ev="http://www.inria.fr/acacia/coswem#"
3   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4   xmlns:cos="http://www.inria.fr/acacia/corese#"
1   xmlns:owl="http://www.w3.org/2002/07/owl#"
2   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
3   xml:base  ="http://www.inria.fr/acacia/coswem#">

4   <!-- Ontology Information -->
5   <owl:Ontology rdf:about="http://www.iria.fr/acacia/CoSWEM-onto.rdfs">
6   <rdfs:label xml:lang="en">EvolutionOntology - Created by Hiep</rdfs:label>
7   <owl:versionInfo xml:lang="fr">Version 1.0 - 2007-04-22</owl:versionInfo>
8   </owl:Ontology>

9   <!-- Description for TraceOnto branch -->
10  <rdfs:Class rdf:ID="TraceOnto">
11  <rdfs:comment xml:lang="en">A trace of ontology changes</rdfs:comment>
12  <rdfs:comment xml:lang="fr">Une trace des changements ontologiques</rdfs:comment>
13  <rdfs:label xml:lang="en">an ontology trace</rdfs:label>
14  <rdfs:label xml:lang="fr">une trace d'ontologie</rdfs:label>
15  </rdfs:Class>

16  <rdf:Property rdf:ID="hasNumber">
17  <rdfs:domain rdf:resource="#TraceOnto"/>
18  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#token"/>
19  <rdfs:comment xml:lang="en">TraceOnto has a number</rdfs:comment>
20  <rdfs:comment xml:lang="fr">TraceOnto a un numero</rdfs:comment>
21  <rdfs:label xml:lang="en">has a number</rdfs:label>
22  <rdfs:label xml:lang="fr">a un numéro</rdfs:label>
23  </rdf:Property>

24  ...
25  </rdf :RDF>
```

Figure 30 - Représentation d'un extrait de l'ontologie d'évolution CoSWEM

Nous présentons dans la Figure 29 un extrait de l'ontologie d'évolution CoSWEM. La première information à modéliser dans cette ontologie est celle des traces de changements effectués dans l'ontologie ainsi que les traces des annotations sémantiques mises à jour, donc nous avons créé le concept `Trace` et ses sous-concepts `TraceOnto` et `TraceAnnot` qui sont équivalents aux traces d'évolution de l'ontologie et de l'annotation sémantique. Pour chaque trace, nous enregistrerons les informations utiles décrivant le processus d'évolution à travers les propriétés de l'ontologie telles que l'auteur qui a réalisé des changements (e.g. *`hasAuthor`*), l'identifiant et la date de la trace effectuée (e.g. *`hasNumber`*, *`hasDate`*), chaque trace conserve les changements effectués entre deux versions de l'ontologie consécutives (e.g. *`hasVersionBefore`*, *`hasVersionAfter`*), etc. La deuxième information importante à modéliser consiste en les changements effectués. Chaque trace contient (e.g. la propriété *`hasTrace`*) plusieurs changements différents (e.g. le concept `Change`).

Nous distinguons aussi des types de changements différents par les concepts classés en hiérarchie, par exemple les changements élémentaires (e.g. `ElementaryChange`), le changement élémentaire sur le concept (e.g. `E_Concept`), l'ajout d'un nouveau concept à l'ontologie (e.g. `InsertConcept`), etc. Les propriétés *`hasChange`*, *`hasAdditionalChange`* sont utilisées pour exprimer les changements supplémentaires à exécuter pour chaque requête de changement (par exemple, si on supprime un concept `c` de l'ontologie, on pourrait avoir envie d'effectuer aussi d'autres changements élémentaires pour rebrancher les sous-concepts de `c` à son super-concept). Comme les traces de l'évolution, nous pouvons également conserver les informations de chaque changement effectué en utilisant les propriétés *`hasCause`*, *`hasChangeNumber`*... ou l'ordre d'exécution des changements par les propriétés *`hasNextChange`* et *`hasPreviousChange`*. Enfin, nous pouvons distinguer les changements qui peuvent causer des inconsistances dans les annotations sémantiques (i.e. correction obligatoire) grâce aux valeurs attribuées de la propriété *`hasAnnotInconsistency`*.

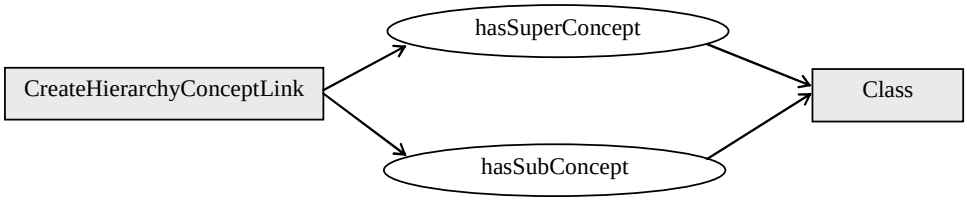


Figure 31 - Un changement ontologique et ses paramètres

En vue de la description des paramètres de chaque changement sur un élément de l'ontologie de domaine, nous utilisons des propriétés pour exprimer ces informations. Par exemple pour le changement `CreateHierarchyConceptLink` *(*c_1, c_2*)* qui vise à créer un lien hiérarchique entre deux concepts c_1 et c_2, nous avons besoin de préciser le super-concept c_2 et le sous-concept c_1 à relier (c.f. Figure 31). Dans ce cas, les propriétés *`hasSuperConcept`* et *`hasSubConcept`* indiquent les relations entre le

changement `CreateHierarchyConceptLink` et les concepts de l'ontologie de domaine par les valeurs (e.g. Class) de ces propriétés.

La Figure 30 représente une partie de l'ontologie d'évolution CoSWEM dans le langage RDF(S) [Brickley et Guha, 2004] [Klyne et Carroll, 2004]. La version complète de cette ontologie d'évolution est présentée dans la partie Annexe B. Nous montrons aussi la représentation des concepts et des propriétés de l'ontologie d'évolution CoSWEM en hiérarchie dans la Figure 32.

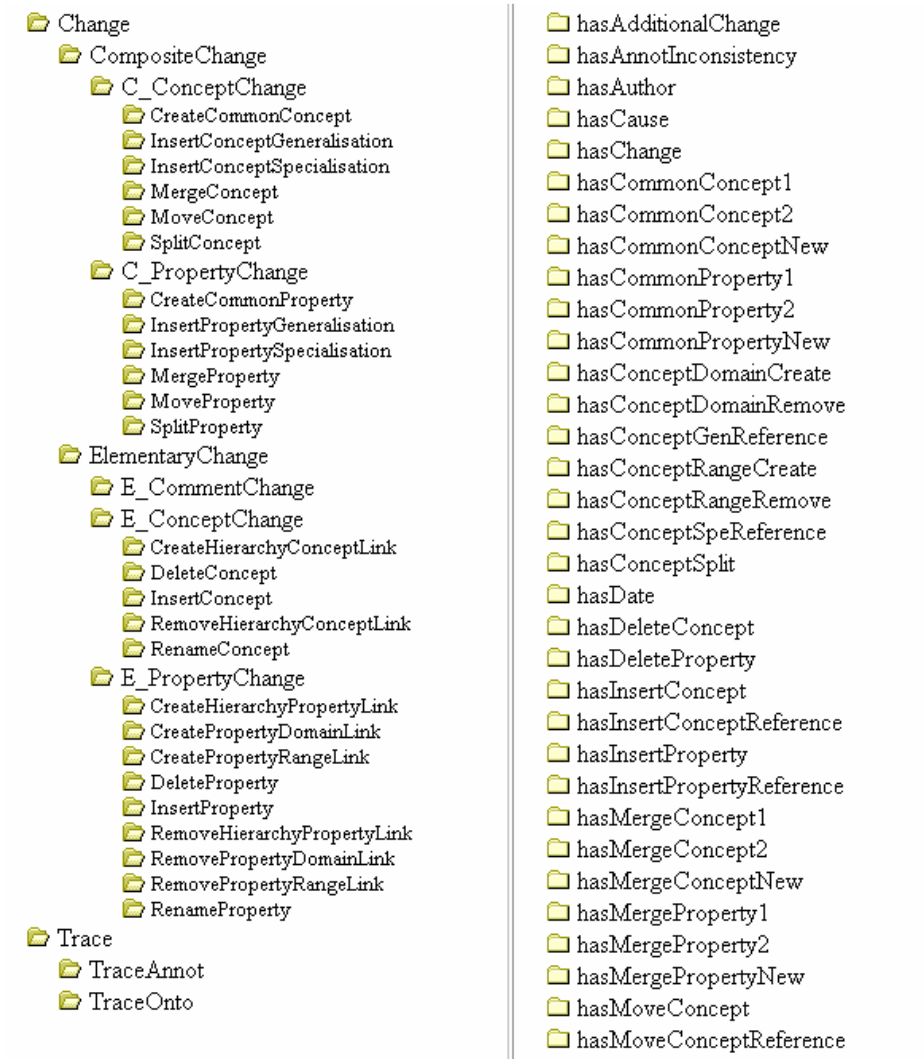


Figure 32 - Les concepts et les propriétées de l'ontologie d'évolution

4.2.3 Trace de changement

Une des tâches importantes du processus de l'évolution est de conserver et de surveiller tous les changements effectués de l'ontologie. Les outils actuels d'édition de l'ontologie offrent aussi cette fonction en fournissant les fichiers journaux ou des logs de versions [Noy et Musen, 2004] [Stojanovic, 2004].

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xml:base="http://www.inria.fr/acacia/coswem#"
xmlns:csw="http://www.inria.fr/acacia/coswem#"
xmlns:ev="http://www.inria.fr/acacia/evolution#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

<csw:TraceOnto rdf:about="http://www.inria.fr/acacia/coswem#TraceO1">
  <csw:hasVersionBefore>Palette-ver1-20070302</csw:hasVersionBefore>
  <csw:hasVersionAfter>Palette-ver1-20073102</csw:hasVersionAfter>
    <csw:hasChange>
          <csw:DeleteConcept
                rdf:about="http://www.inria.fr/acacia/evolution#id-
                deleteconcept01">
            <csw:hasDeleteConcept rdf:resource="http://www.inria.fr/acacia/evolution#engineer-
            student"/>
          <csw:hasAnnotInconsistency>yes</csw:hasAnnotInconsistency>
          <csw:hasAdditionalChange>
            <csw:CreateHierarchyConceptLink
                rdf:about="http://www.inria.fr/acacia/coswem#id-
                createhierarchyconceptlink01">
                <csw:hasSubConceptCreate
              rdf:resource="http://www.inria.fr/acacia/evolution#ITengineer_student"/>
                <csw:hasSuperConceptCreate
          rdf:resource="http://www.inria.fr/acacia/evolution#Student"/>
            </csw:CreateHierarchyConceptLink>
          </csw:hasAdditionalChange>
<.....>
          <csw:hasAdditionalChange>
            <csw:CreateConceptDomainLink
                rdf:about="http://www.inria.fr/acacia/coswem#id-
                createconceptdomainlink01">
                <csw:hasConceptDomainCreate
                rdf:resource="http://www.inria.fr/acacia/evolution#PhD_Student"/>
                <csw:hasPropertyDomainCreate
                rdf:resource="http://www.inria.fr/acacia/evolution#studyIn"/>
            </csw:CreateConceptDomainLink>
          </csw:hasAdditionalChange>
<.....>
      </csw:DeleteConcept>
    </csw:hasChange>
    <csw:hasChange>
          <csw:MergeConcept
                rdf:about="http://www.inria.fr/acacia/coswem#id-
                merge01">
                <csw:hasMergeConcept1
          rdf:resource="http://www.inria.fr/acacia/evolution#Master1_Student"/>
                <csw:hasMergeConcept2
          rdf:resource="http://www.inria.fr/acacia/evolution#Master2_Student"/>
          <csw:hasMergeConceptNew
          rdf:resource="http://www.inria.fr/acacia/evolution#Master_Student"/>
          </csw:MergeConcept>
    </csw:hasChange>
</csw:TraceOnto>
</rdf:RDF>
```

Figure 33 - Extrait d'une trace de changements

Dans notre travail, nous utilisons également des fichiers de journaux d'évolution qui nous permettent de capturer les changements exécutés ainsi que la trace de changements (ou la trace d'évolution) et l'historique des modifications

sur l'ontologie. Nous représentons la trace de changements sous forme d'une annotation sémantique qui utilise des termes définis dans l'ontologie d'évolution CoSWEM. La Figure 33 représente l'extrait d'une trace d'évolution de l'ontologie (i.e.`TraceOnto`). Cette trace d'évolution capture des changements effectués entre deux versions ontologiques consécutives `Palette-ver1-20070302` et `Palette-ver1-20073102`. Nous trouvons dans cette trace d'évolution les changements effectués via des relations *<csw:hasChange>* tels que la suppression du concept (i.e. `DeleteConcept`) et la fusion des concepts (i.e. `MergeConcept`). Un changement de l'ontologie à effectuer peut provoquer d'autres changements additionnels, nous les détectons par les relations *<csw:hasAdditionalChange>*. D'autre part, les informations des entités changées de l'ontologie de domaine sont aussi conservées dans cette trace d'évolution. Par exemple, le concept supprimé `engineer-student` est relié avec le changement `<csw:DeleteConcept>` par la propriété *<csw:hasDeleteConcept>***.** De la même façon, toutes les informations des autres entités concernant les changements effectués sont enregistrées et serviront au processus d'évolution.

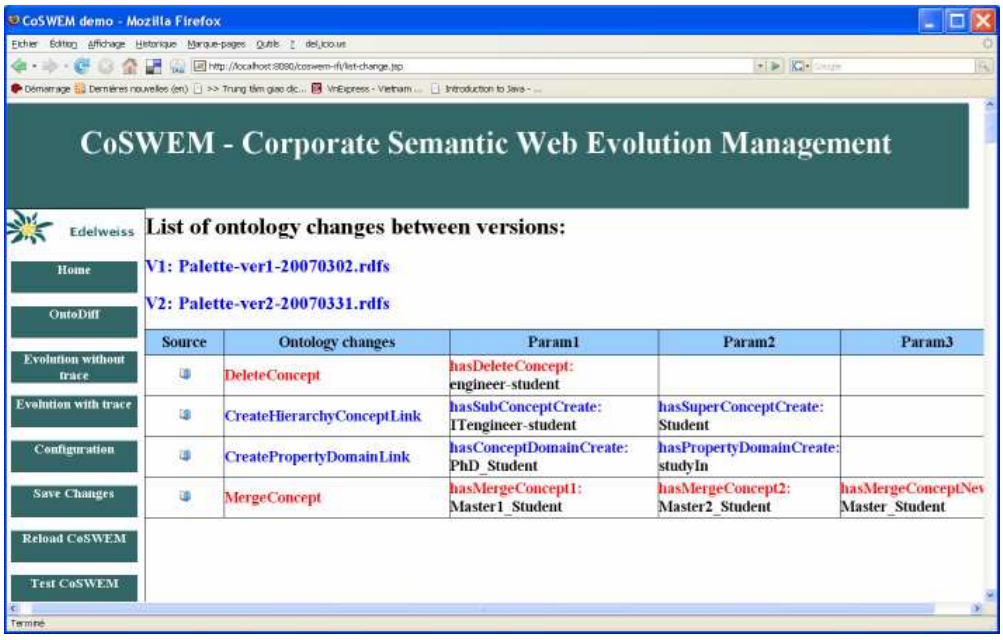


Figure 34 - Exemple d'une trace de changement de l'ontologie

Cette trace d'évolution est également représentée dans les interfaces graphiques du système CoSWEM (c.f. Figure 34). Les changements ontologiques effectués sont visualisés avec leurs paramètres qui sont les entitées modifiées de l'ontologie de domaine. Cette interface permet à l'utilisateur de surveiller les changements exécutés ainsi que les éléments affectés par chaque changement ontologique. Dans cette interface, on peut aussi distinguer les types de changements ontologiques entraînant la correction obligatoire (i.e. changements en couleur rouge) et les changements entraînant la correction facultative (i.e. changements en couleur bleue).

4.3 Résolution des changements de l'ontologie

4.3.1 Niveau d'inconsistance

[Gruber, 1993] définit *une ontologie comme une spécification explicite d'une conceptualisation*. Une *conceptualisation* est une abstraction du monde que nous souhaitons représenter dans un certain but. La *conceptualisation* est le résultat d'une analyse ontologique du domaine étudié. L'ontologie est une *spécification* parce qu'elle représente la conceptualisation dans une forme concrète. Elle est *explicite* parce que tous les concepts et les contraintes utilisés sont explicitement définis. En reposant sur cette définition, [Klein, 2004] fait également une distinction entre les différents niveaux d'interprétation :

- *Changement conceptuel*: un changement dans la conceptualisation ;
- *Changement de spécification:* un changement des spécifications d'une conceptualisation ;
- *Changement de représentation:* un changement de la représentation des spécifications d'une conceptualisation.

Chaque niveau de changement peut impliquer un type d'inconsistance différente. En effet, ces trois niveaux d'interprétation ci-dessus de l'ontologie peuvent être considérés équivalents à trois niveaux d'inconsistance : (i) *niveau sémantique*, (ii) *niveau structurel* et (iii) *niveau syntaxtique.* [Davies et al., 2002] distingue aussi les schémas RDFS et les documents RDF en trois *niveaux d'abstraction* correspondant à ces derniers.

Outre les recherches sur l'inconsistance de structure, il existe également des recherches sur l'inconsistance logique. Alors que la consistance structurelle assure seulement que l'ontologie reste conforme à certaines contraintes structurelles (e.g. utilisation conforme du concept, de la propriété…), la consistance logique assure que l'ontologie est "sémantiquement consistante", c'est-à-dire qu'elle ne contient pas des informations contradictoires. Concernant la consistance logique, [Haase et al., 2005] définit une interprétation ***I*** qui contient des ensembles des éléments du modèle de l'ontologie et des fonctions d'interprétation. Une interprétation ***I*** satisfait une ontologie O si elle satisfait les conditions sémantiques (e.g. des axiomes) de cette interprétation. Nous avons présenté dans le chapitre 2 certains travaux existants qui abordent le sujet des inconsistances de l'ontologie au niveau sémantique tels que les recherches sur la résolution des inconsistances de [Flouris, 2006] et [Plesser, 2006]. Selon ces approches, une ontologie est considérée incompatible (ou inconsistante) si et seulement si il n'y a aucune interprétation satisfaisant tous les axiomes en logique de description de cette ontologie.

Dans ce mémoire, nous nous intéressons à l'affectation des changements ontologiques sur les annotations sémantiques utilisant les instances ou les types de

concepts et de propriétés définis dans l'ontologie. Par conséquent, ce travail est en relation étroite avec les niveaux d'inconsistance de structure. Nous allons nous focaliser sur le niveau structurel dans notre processus de l'évolution de l'ontologie. La section suivante présente les stratégies développées, respectivement pour chacun des types de changement, et dont l'objectif est de résoudre les inconsistances générées par ledit changement.

4.3.2 Stratégies de résolution

Pour l'évolution de l'ontologie, les ontologistes ont des points de vue divergents sur la manière dont un changement ontologique est effectué. Ces vues sont affectées par la connaissance, les préférences ou les idées personnelles sur le domaine… de l'ontologiste. Un changement ontologique est donc implémenté selon différentes manières, chacune entraînera l'ontologie vers une autre ontologie consistante différente.

D'autre part, les ontologies actuelles deviennent plus variées et complexes. Elles sont habituellement développées par plusieurs ontologistes. Dans les chapitres précédents, nous avons mentionné qu'un changement d'une part de l'ontologie pourrait affecter la consistance d'autres parties de la même ontologie [Stojanovic et al., 2003]. L'ontologiste qui a fait un changement ontologique peut ignorer certains effets de ce changement, car il pourrait ne pas connaître toutes les parties de l'ontologie. Cela peut affecter la consistance de toute l'ontologie. Par conséquent, nous avons besoin d'un mécanisme de gestion des manières d'exécution des changements ontologiques. Ce mécanisme aidera l'ontologiste à choisir la solution "la plus convenable" pour effectuer un changement de l'ontologie. Pour ce faire, nous proposons des *stratégies de résolution* qui permettent de résoudre les changements de l'ontologie d'une manière automatique ou avec l'intervention de l'ontologiste en vue d'assurer la consistance de toutes les parties de l'ontologie après sa modification.

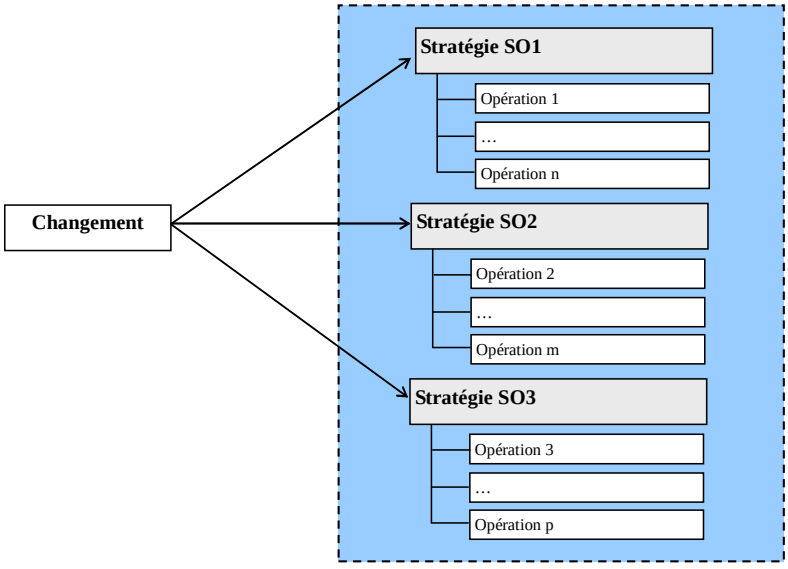


Figure 35 - Stratégies de résolution pour l'ontologie

Puisque chaque changement ontologique peut être effectué selon différentes manières, par exemple dans la Figure 35, un changement à résoudre peut être équivalent à trois stratégies différentes (c.f. SO1, SO2, SO3), chaque stratégie correspond à un ensemble de changements additionnels différents à effectuer (c.f. opération 1, …, opération n). Cela veut dire que les différents ensembles de changements additionnels seraient également générés pour compléter le changement ontologique initial. Revenons à l'exemple d'un extrait de l'ontologie O'CoP (c.f. Figure 12, section 2.2.2), si nous supprimons le concept `Member_role`, nous pourrions avoir les différentes manières suivantes pour traiter les sous-concepts de `Member_role` : (i) ses sous-concepts (c.f. les concepts `Facilitator` et `Coordinator`) sont supprimés ou (ii) ses sous-concepts sont rebranchés à son super-concept `Role_in_the_CoP`. L'ontologiste va choisir une solution parmi celles proposées pour appliquer au type de changement de suppression d'un concept. Nous trouverons aussi les stratégies de résolution `SO-12` et `SO-13` qui sont équivalentes au changement "suppression d'un concept au milieu" dans le Tableau 7.

Afin d'aider l'ontologiste dans le travail de modification de l'ontologie, il est souhaitable de lui proposer une bibliothèque de stratégies de résolution dans laquelle il choisira celle qui lui convient. Nous avons construit toutes les stratégies de résolution correspondants aux changements ontologiques classées dans notre travail. Le Tableau 7 représente les stratégies proposées pour le changement de suppression d'un concept de l'ontologie. Ces stratégies couvrent les cas qui peuvent se produire pour ce changement de suppression tels que : le concept à supprimer est un concept feuille, ou il s'agit d'un concept milieu sans connexion avec des propriétés, ou il s'agit d'un concept milieu qui est le domaine/co-domaine d'une propriété, etc. La bibliothèque complète des stratégies de résolution des changements de l'ontologie est présentée dans la partie Annexe A.

Tableau 7 - Stratégies de résolution pour le changement DeleteConcept

<table>
<tr><td rowspan="2">Changement</td><td colspan="2"><u>Syntaxe :</u> <code>DeleteConcept(c)</code></td></tr>
<tr><td colspan="2"><u>Sémantique :</u> Supprimer le concept <code>c</code></td></tr>
<tr><td rowspan="2">DeleteConcept</td><td colspan="2"><u>Pré-condition :</u> $c \in \mathbf{C}$</td></tr>
<tr><td colspan="2"><u>Post-condition :</u> $c \notin \mathbf{C}$</td></tr>
<tr><td>a. Cas de changement</td><td colspan="2">b. Stratégie de résolution: pour les éléments dépendants de l'ontologie</td></tr>
<tr><td rowspan="3">1.2.1 - Si <code>c</code> est un concept feuille, possédant un seul père c_2, et sans lien avec aucune propriété.
$c, c_2 \in \mathbf{C}, (c, c_2) \in \mathbf{H_c}$</td><td rowspan="2">SO-2</td><td>Traitement des instances du concept <code>c</code>
- Supprimer les instances du concept <code>c</code></td></tr>
<tr><td>- Attacher les instances du concept <code>c</code> au concept père c_2 de <code>c</code></td></tr>
<tr><td>SO-3</td><td>- Supprimer le lien entre le concept <code>c</code> et son concept père c_2</td></tr>
</table>

$\neg\exists c_1 \in \mathbf{C}:(c_1 \neq c_2) \wedge (c, c_1) \in \mathbf{H_c}$ $\neg\exists c_0 \in \mathbf{C}:(c_0, c) \in \mathbf{H_c}$ $\neg\exists p \in \mathbf{P}: c \in$ domain (p) ou $c \in$ range(p)		- Supprimer le concept c
1.2.2 - Si c est un concept feuille, possédant un seul père, avec c appartenant au domaine/co-domaine d'une propriété *p*. $c, c_2 \in \mathbf{C}, p \in \mathbf{P}$ $(c, c_2) \in \mathbf{H_c}$ $\neg\exists c_1 \in \mathbf{C}: c_1 \neq c_2) \wedge (c, c_1) \in \mathbf{H_c}$ $\neg\exists c_0 \in \mathbf{C}:(c_0, c) \in \mathbf{H_c}$ $c \in$ domain (p) ou $c \in$ range(p)	**SO - 4**	Traitement des instances du concept c *Idem SO - 2*
	SO - 5	- Supprimer le lien entre le concept c et son concept père c_2 - Supprimer le lien entre c et la propriété *p* - Supprimer le concept c
1.2.3 - Si c est un concept feuille, possédant plusieurs pères, avec c appartenant au domaine/co-domaine d'une propriété *p* $c, c_1, c_2 \in \mathbf{C}, p \in \mathbf{P}$ $(c, c_1) \in \mathbf{H_c}, (c, c_2) \in \mathbf{H_c}$ $\neg\exists c_0 \in \mathbf{C}:(c_0, c) \in \mathbf{H_c}$ $c \in$ domain (p) ou $c \in$ range(p)	**SO - 6**	Traitement des instances du concept c *Idem SO - 2*
	SO - 7	- Supprimer le lien entre le concept c et tous ses concept pères de c (e.g. c_1, c_2…) - Supprimer le lien entre c et la propriété *p* - Supprimer le concept c
1.2.4 – Si c est un concept milieu, possédant un seul père, sans lien avec aucune propriété. $c_0, c, c_1, c_2 \in \mathbf{C}$ $(c_0, c) \in \mathbf{H_c}, (c, c_2) \in \mathbf{H_c}$ $\neg\exists c_1 \in \mathbf{C}:(c, c_1) \in \mathbf{H_c}$ $\neg\exists p \in \mathbf{P}: c \in$ domain (p) ou $c \in$ range(p)	**SO - 8**	Traitement des instances du concept c *Idem SO - 2*
	SO – 9	- Supprimer le lien entre le concept c et son concept père c_2 - Supprimer le lien entre c et ses concepts fils c_i - Supprimer le concept c - Supprimer les concepts fils c_i du concept c *Note : Traiter ensuite les concepts fils c_i (s'il y en a) du concept* c

	SO - 10	- Supprimer le lien entre le concept c et son concept père c_2 - Supprimer le lien entre le c et ses concepts fils c_i - Supprimer le concept c - Rebrancher les concepts fils c_i du concept c au concept père c_2 *Note : Traiter ensuite les concepts fils* c_i *(s'il y en a) du concept* c
1.2.5 – Si c est un concept milieu, possédant un seul père, avec c appartenant au domaine/co-domaine d'une propriété *p* $c_0, c, c_1, c_2 \in \mathbf{C}$ $p \in \mathbf{P}$ $(c_0, c) \in \mathbf{H_c}$ $(c, c_2) \in \mathbf{H_c}$ $\neg\exists c_1 \in \mathbf{C}:(c_1 \neq c_2) \wedge (c, c_1) \in \mathbf{H_c}$ $c \in$ domain (p) ou $c \in$ range(p)	**SO - 11**	Traitement des instances du concept c *Idem SO - 2*
	SO – 12	- Supprimer le lien entre le concept c et son concept père c_2 - Supprimer le lien entre c et ses concepts fils c_i - Supprimer le lien entre c et la propriété p - Supprimer le concept c - Supprimer les concepts fils c_i du concept c *Note : Traiter ensuite les concepts fils* c_i *(s'il y en a) du concept* c
	SO – 13	- Supprimer le lien entre le concept c et son concept père c_2 - Supprimer le lien entre c et ses concepts fils c_i - Supprimer le lien entre c et la propriété p - Supprimer le concept c - Attacher les concepts fils c_i du concept c au concept père c_2 *Note : Traiter ensuite les concepts fils* c_i *(s'il y en a) du concept* c
1.2.6 – Si c est un concept milieu, possédant plusieurs pères, avec c appartenant au domaine/co-domaine d'une propriété *p* $c_0, c, c_1, c_2 \in \mathbf{C}, p \in \mathbf{P}$ $(c_0, c) \in \mathbf{H_c}, (c, c_1) \in \mathbf{H_c}$ $(c, c_2) \in \mathbf{H_c}$ $c \in$ domain (p) ou $c \in$ range(p)	**SO - 14**	Traitement des instances du concept c *Idem SO - 2*
	SO - 15	- Supprimer le lien entre le concept c et tous ses concepts pères (e.g. c_1, c_2) - Supprimer le lien entre c et ses concepts fils c_i - Supprimer le lien entre c et la propriété p - Supprimer le concept c - Supprimer les concepts fils c_i du concept c *Note : Traiter ensuite les concepts fils* c_i *(s'il y en a) du concept* c

	SO - 16	- Supprimer le lien entre le concept c et tous ses concepts pères (e.g. c_1, c_2) - Supprimer le lien entre c et ses concepts fils c_i - Supprimer le lien entre c et la propriété p - Supprimer le concept c - Attacher les concepts fils c_i du concept c au concept père quelconque c_x indiqué par l'utilisateur *Note : Traiter ensuite les concepts fils c_i (s'il y en a) du concept c*

4.3.3 Résolution des inconsistances de l'ontologie

La tâche de la phase de résolution des inconsistances est de résoudre des changements ontologiques d'une manière systématique, assurant la consistance de toute l'ontologie. Cette phase peut être réalisée en complétant les changements exigés avec un ensemble de changements additionnels, qui garantit le transfert d'une ontologie consistante initiale à un autre état consistant.

Figure 36 - Interface de l'édition de l'ontologie dans l'éditeur ECCO

Dans le cadre de cette thèse, nous n'avons pas l'ambition de développer un éditeur d'ontologie permettant de résoudre des changements de l'ontologie selon

les requêtes de l'utilisateur. Parce qu'il existe déjà maintenant certains outils qui fournissent des fonctions principales par rapport à l'édition d'une ontologie telles que l'ajout, la suppression, le renommage d'une entité de l'ontologie. En outre, un outil d'édition de l'ontologie n'est pas un des objectifs à atteindre dans notre travail, donc l'idée de réutiliser un outil d'édition nous semblait parfaitement convenable.

En effet, nous réutilisons l'outil d'édition de l'ontologie ECCO[38] (c.f. Figure 22, section 3.5.2) développé par les collègues de notre équipe. Cet outil aide au processus de création et de modification d'une ontologie. La Figure 36 présente une interface de l'outil ECCO dans laquelle nous trouverons les fonctionnalités offertes par ECCO telles que la création, la suppression, le changement de label, le changement d'ordre des entités (i.e. les concepts et les propriétés) de l'ontologie. Cet outil fournit des interfaces graphiques qui sont faciles à utiliser, par exemple il nous permet de visualiser les concepts et les propriétés d'une manière hiérarchique et nous permet aussi de changer des liens hiérarchiques des entités de l'ontologie en déplaçant les éléments par des opérations *drag-and-drop* (c.f. Figure 37).

Figure 37 - Hiérarchie de concepts dans l'éditeur ECCO

Au cours du processus de modification de l'ontologie, l'outil ECCO va capturer une trace des changements ontologiques effectués entre deux versions de l'ontologie. Ensuite, il va générer automatiquement la trace d'évolution sous forme d'une annotation sémantique. Cette trace nous facilitera la recherche et la résolution des annotations sémantiques affectées par les changements effectués. Nous allons présenter en détail ce processus dans le chapitre suivant.

Dans les perspectives à court terme, nous allons implémenter dans cet outil ECCO les stratégies de résolution pour les types de changements de l'ontologie. Nous avons besoin d'un mécanisme pour résoudre des changements ontologiques d'une manière systématique et automatique parce que si cela est laissé à la charge d'un ontologiste, le processus d'évolution risque d'être trop inexact et long, il est peu réaliste de s'attendre à ce que les humains puissent comprendre toute l'ontologie et ses interdépendances.

[38] Editeur Collaboratif et Contextuel d'Ontologie

4.3.4 Gestion et comparaison des versions de l'ontologie

Dans l'état de l'art, nous avons distingué entre l'évolution de l'ontologie et la gestion des versions de l'ontologie (versioning). La différence principale est : l'évolution de l'ontologie facilite la modification d'une ontologie en préservant son état de consistance tandis que la gestion des versions de l'ontologie permet la manipulation des changements de l'ontologie en créant et en gérant ses différentes versions. Un aspect important d'une méthodologie de gestion de changement est la capacité de comparer des versions des ontologies et de souligner les différences au niveau structurel. La comparaison doit trouver les changements effectués entre deux versions de l'ontologie même si ceux-ci se sont produits d'une manière non contrôlée ou dans un ordre inconnu.

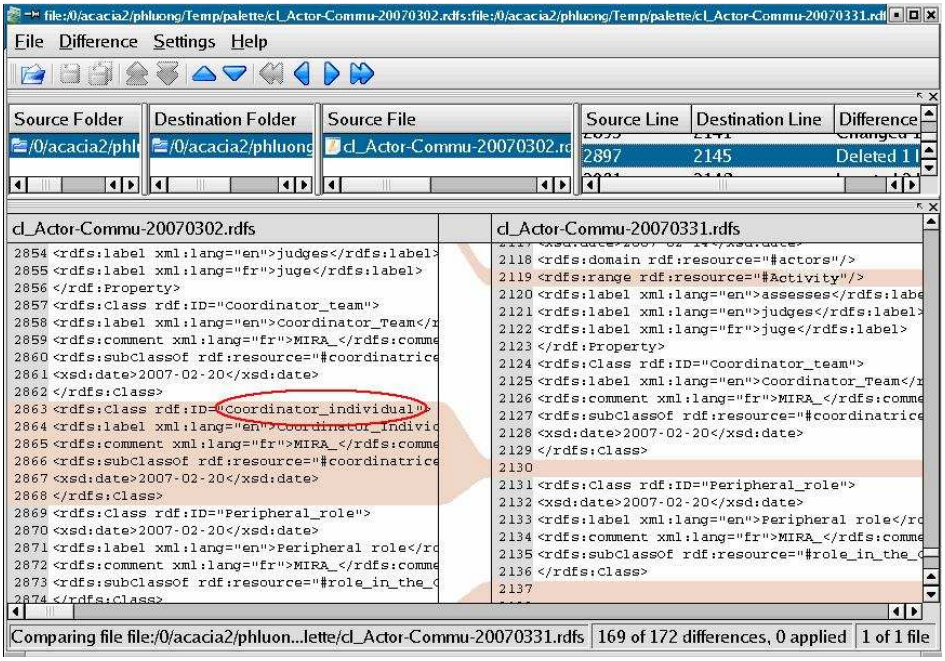


Figure 38 - Comparaison des lignes de texte supprimés entre deux versions

Dans notre système de gestion de l'évolution CoSWEM, une fonction de comparaison des différences entre deux versions de l'ontologie est implémentée. Cette fonction a l'avantage de détecter les concepts ou les propriétés changées. Nous pouvons chercher les entités qui existent dans une version mais n'existent pas dans l'autre. Dans le cas où on possède plusieurs informations (i.e. la trace de changement entre deux versions de l'ontologie), cette fonction peut préciser quel type de changement a été effectué et comment les entités concernées sont modifiées, etc.

Des outils d'édition offrant la fonction de comparaison des différents morceaux de texte peuvent marquer seulement les parties ajoutées ou enlevées entre deux versions selon l'ordre des lignes de texte. Cela est très difficile pour surveiller les changements. En outre, nous ne pouvons pas bien regarder la

relation entre les entités différentes par rapport aux autres éléments existants dans l'ontologie (e.g. la relation hiérarchique entre des concepts). Par exemple, dans la Figure 38, l'interface de l'outil de texte (i.e. outil Kompare sous Linux) représente la suppression de la classe `Coordinator_individual` mais nous ne pouvons pas savoir quelle est la relation de cette classe modifiée avec les classes dans la même branche (e.g. quelle est la super-classe, la sous-classe, etc.)

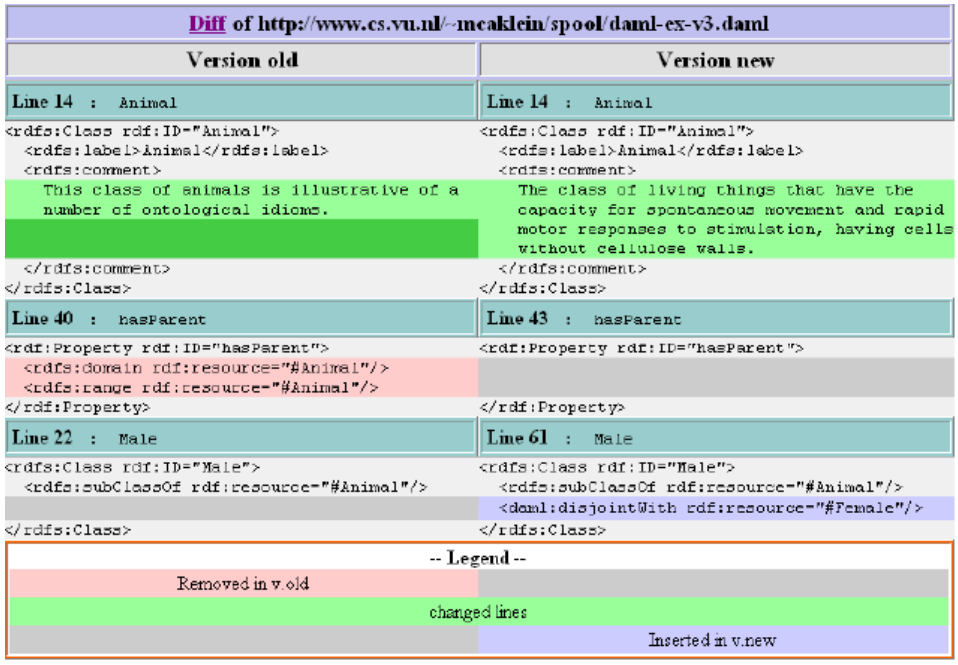


Figure 39 - Comparaison des versions de l'ontologie de l'outil OntoView

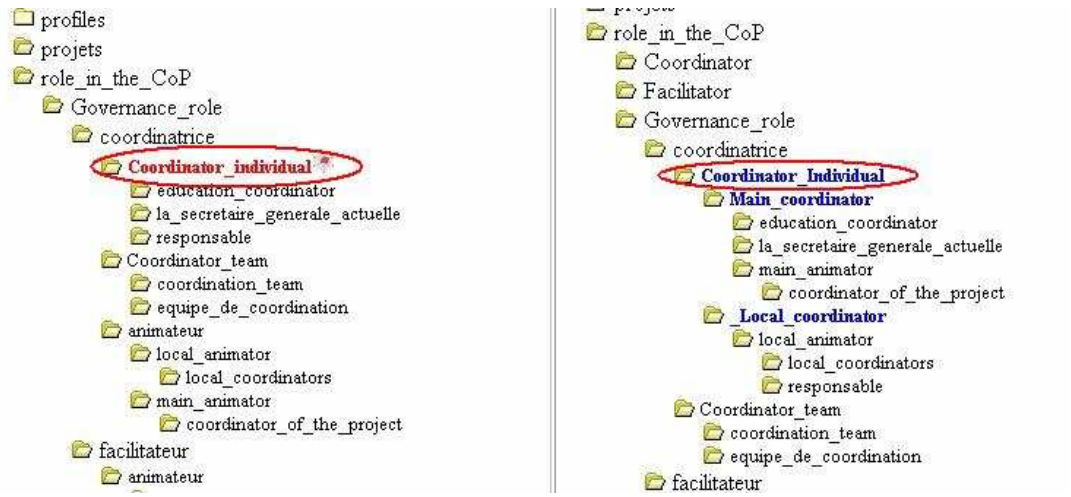


Figure 40 - Comparaison des concepts différents entre deux versions de l'ontologie par l'outil CoSWEM

Dans notre outil CoSWEM, cette fonction a été améliorée. La Figure 40 représente une comparaison des concepts différents entre deux versions de l'ontologie. La colonne à gauche représente l'ancienne version de l'ontologie et celle à droite la nouvelle version de l'ontologie. A cette étape, nous pouvons trouver les différences au niveau des entités, particulièrement le concept en rouge `Coordinator_individual` (c.f. Figure 40) exprime que ce concept a été modifié et il n'apparaît pas dans la nouvelle version. En revanche, les concepts en bleu `Coordinator_Individual`, `Main_coordinator`, `_Local_coordiantor` sont ajoutés dans la nouvelle version et ils n'apparaissent pas dans l'ancienne version de l'ontologie.

Concernant cette fonction, [Klein, 2004] présente aussi l'outil de comparaison des versions de l'ontologie OntoView (c.f. Figure 39). Dans cet outil, les auteurs comparent les versions ontologiques au niveau structurel, ils montrent quels concepts ou quelles propriétés ont été changées entre deux versions. Les éléments changements sont représentés avec des couleurs différentes (c.f. Figure 40). Cependant, cette approche dépend toujours du fichier journal capturant des changements exécutés entre deux versions de l'ontologie. Notre outil CoSWEM apporte une nouveauté sur ce point, on peut toujours avoir une vue générale sur les différences entre deux versions même dans le cas où on ne garde plus la trace de changement entre les versions de l'ontologie.

D'autre part, nous intégrons le moteur de recherche sémantique Corese [Corby et al., 2006] dans le module de comparaison pour effectuer des requêtes sur les ontologies. Cette intégration nous permet de faire une recherche des entités ontologiques plus exacte et plus efficace car le moteur de recherche Corese fait une recherche de manière *sémantique* et non pas *syntaxique* comme les outils de comparaison existants. Ce moteur de recherche nous permet de charger des ontologies représentées dans le langage RDFS et de faire des recherches d'information sur les hiérarchies de concepts et de propriétés des ontologies ainsi que sur la base d'annotations qui utilise des termes définis dans ces ontologies. Afin de comparer les différentes versions de l'ontologie, nous avons crée des différentes instances indépendantes de Corese qui ont pour but de réaliser des tâches spécifiques. Par exemple, l'instance `corese_1` chargera l'ancienne version de l'ontologie dans le système permettant d'interroger des requêtes sur son hiérarchie de concepts et de propriété tandis que la deuxième instance `corese_2` vise à faire des requêtes sur la nouvelle version de l'ontologie. En comparant les résultats envoyés par Corese sur ces deux versions de l'ontologie, nous pouvons déterminer les éléments différents au niveau de structure (i.e. les concepts et les propriétés différents) entre ces versions. Dans le cas où la trace de changement est gardée, CoSWEM peut trouver exactement les entités modifiées, les types de changements effectués ainsi que les annotations sémantiques affectées par ces changements ontologiques. Il nous rend également plus d'information sur les changements effectués. Par exemple, nous trouvons certaines différences entre deux versions de l'ontologie dans la Figure 41 . Non seulement ces différences sont distinguées par les couleurs différentes mais on peut savoir comment les changements ont été réalisés grâce à la trace de changement. Nous pouvons

montrer que le concept `ACTIVITY` dans l'ancienne version a été renommé en `Activity` dans la nouvelle version.

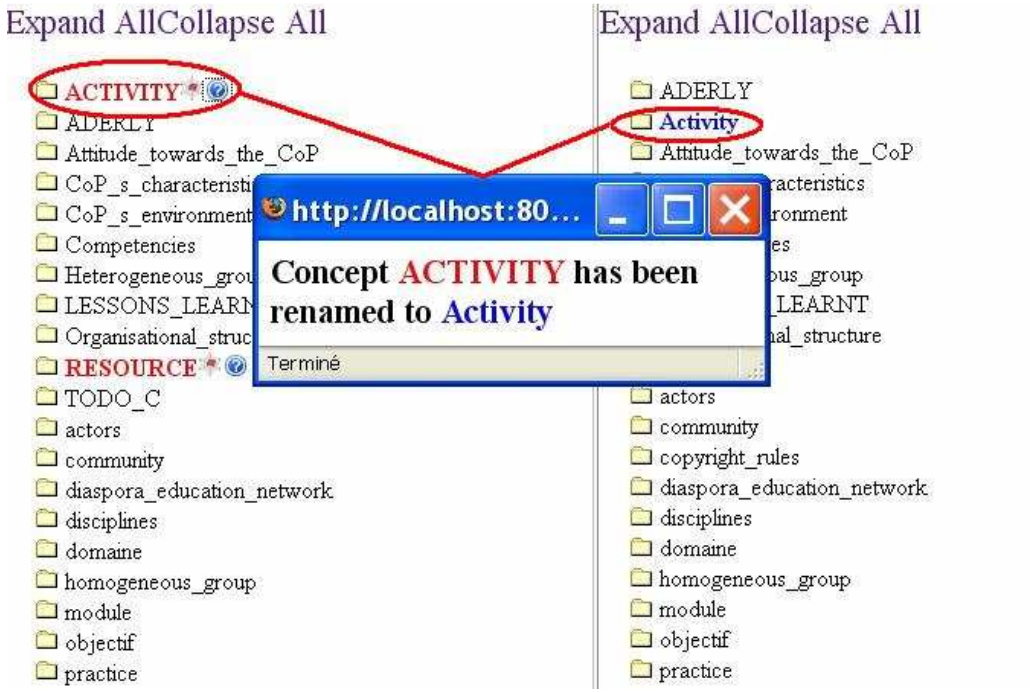


Figure 41 - Comparaision des versions de l'ontologie dans CoSWEM

4.4 Propagation des changements de l'ontologie

La tâche de la propagation du changement du processus d'évolution de l'ontologie doit assurer la consistance des parties dépendantes après avoir exécuté une mise à jour d'ontologie. Ces parties dépendantes peuvent inclure des ontologies dépendantes, des instances, ainsi que des programmes d'application utilisant l'ontologie. Les recherches existantes s'intéressent aux trois types de propagation des changements de l'ontologie ci-dessous [Stojanovic, 2004]:

- L'impact des changements sur les ontologies dépendantes : une mise à jour d'ontologie pourrait corrompre d'autres ontologies qui dépendent de l'ontologie modifiée. Ces ontologies sont construites à partir de l'ontologie modifiée ou elles l'importent. Ce problème peut être résolu par une procédure récursive en appliquant des changements sur ces ontologies afin de préserver leur consistance conceptuelle, structurelle et comportementale.

- L'impact des changements sur les instances ontologiques : quand l'ontologie est modifiée, les instances doivent être changées pour se conformer à l'ontologie modifiée. Cela signifie que l'adaptation continue des informations annotées à une nouvelle terminologie sémantique est nécessaire.

- L'impact des changements sur les applications : les changements de l'ontologie pourront influencer les applications qui fonctionnent sur l'ontologie modifiée et les bases de connaissance.

4.4.1 Propagation aux ontologies dépendantes

Il existe actuellement des outils d'édition de l'ontologie qui permettent de construire une ontologie à partir de thésaurus ou de corpus de texte, c-à-d. ces outils nous supportent de créer l'ontologie à partir de zéro. Cependant, cette manière n'est pas très efficace et fait perdre du temps. En revanche, on peut réutiliser des composants ou des ontologies de base qui sont bien développés. Une ontologie peut inclure des concepts ou des propriétés définies dans une autre ontologie par le mécanisme de modularisation [Stojanovic, 2004]. Cette ontologie est appelé l'ontologie dépendante qui repose sur une ontologie de base. La Figure 42 présente un exemple de réutilisation des ontologies ainsi que des ontologies dépendantes. L'ontologie de base modifiée pourrait entraîner les inconsistances sur les ontologies dépendantes O1 et O2.

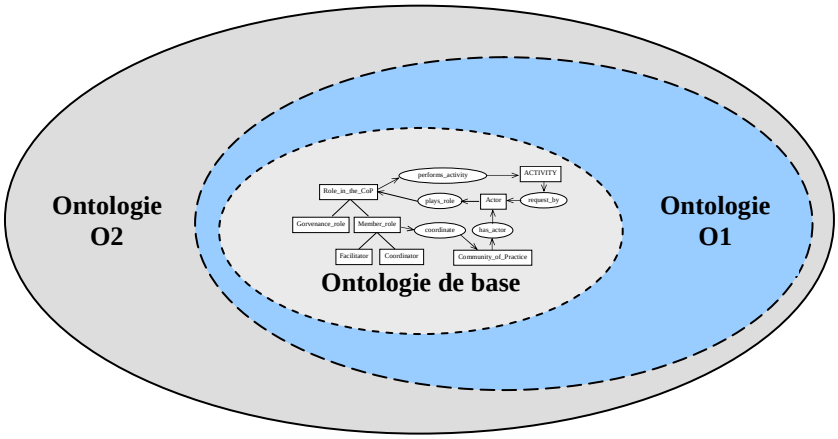


Figure 42 - Réutilisation des ontologies et les ontologies dépendantes

La propagation des changements vers les ontologies dépendantes (i.e. les ontologies incluant l'ontologie de base qui a été modifiée) est un des sujets importants dans les recherches de notre domaine mais nous n'avons pas l'ambition de le résoudre dans le cadre de cette thèse. Ce travail sera implémenté dans l'outil d'édition ECCO développé par notre équipe.

4.4.2 Propagation aux annotations sémantiques

Dans notre travail, nous nous intéressons à la propagation des changements de l'ontologie vers les annotations sémantiques concernées qui utilisent les termes définis (i.e. les concepts, les propriétés) de l'ontologie. Après avoir modifié

l'ontologie, ces annotations pourraient devenir inconsistantes à cause de certains changements ontologiques effectués.

En vue de résoudre des annotations sémantiques inconsistantes, nous proposons aussi des stratégies de résolution pour les corriger. Puisque les changements sont propagés à partir du processus de l'évolution, nous construisons donc un ensemble de stratégies qui sont équivalentes à chaque changement ontologique. Autrement dit, associé à chaque changement ontologique, nous avons un ensemble de stratégies de résolution pour corriger les inconsistances de l'ontologie et des annotations sémantiques (c.f. Figure 43). Nous allons présenter particulièrement la propagation des changements ontologiques vers les annotations sémantiques ainsi que l'évolution de l'annotation dans le chapitre suivant.

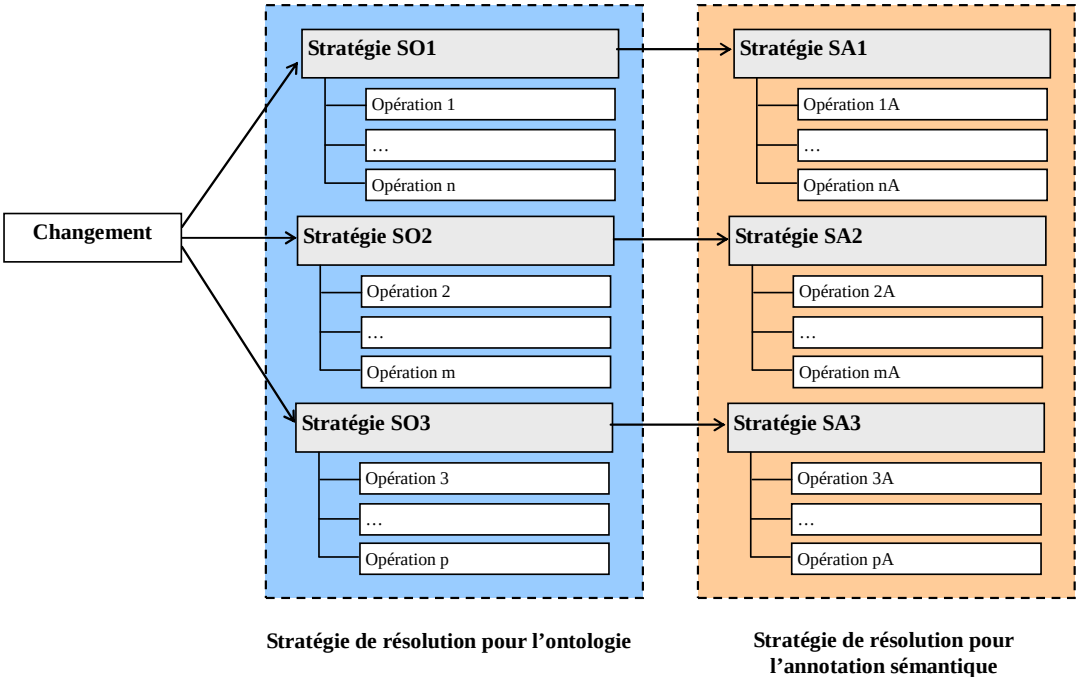


Figure 43 - Propagation du changement de l'ontologie vers l'annotation sémantique

4.5 Conclusion

Dans ce chapitre, nous avons présenté un des composants principaux du système de gestion de l'évolution CoSWEM, à savoir l'évolution de l'ontologie. La tâche importante du processus de l'évolution de l'ontologie est d'assurer la consistance de toute l'ontologie et de ses parties dépendantes après avoir effectué des changements sur l'ontologie. Nous avons tout d'abord défini un modèle formel de l'ontologie et classé l'ensemble des changements de l'ontologie selon le niveau d'abstraction ou leur affectation sur les annotations sémantiques. En reposant sur cette classification des changements, nous avons établi une ontologie

d'évolution qui sert à modéliser les types de changements effectués au cours de l'évolution.

Pour développer l'ontologie d'évolution, nous utilisons l'outil d'édition de l'ontologie ECCO développé par notre équipe. Cet outil fournit plusieurs fonctions nécessaires qui facilitent la création ou l'édition d'une ontologie selon les besoins d'un ontologiste. Nous ne nous sommes pas concentrés dans ce chapitre sur le travail de résolution des changements (ou la sémantique de changement) car cela ne figurait pas dans les objectifs principaux de nos travaux de thèse. Cependant, nous avons construit une bibliothèque complète de stratégies de résolution des changements ontologiques qui permettent de réaliser des changements d'une manière systématique et automatique. Ces stratégies guident l'ontologiste dans la modification et le choix de la solution la plus convenable pour chaque type de changement. Dans une perspective à court terme, nous sommes en train d'implémenter ces stratégies de résolution dans l'outil d'édition ECCO en vue de construire un éditeur d'ontologie puissant et efficace pour traiter les ontologies tout au long du processus de l'évolution.

Nous avons également présenté un avantage de notre processus d'évolution de l'ontologie qui est la capacité de gestion des versions et de comparaison des différentes versions d'une ontologie. Cette fonction permet une comparaison très efficace et exacte entre les variantes de l'ontologie car nous avons appliqué le moteur de recherche sémantique Corese dans le but de comparer les structures des versions de l'ontologie. Cette fonctionnalité offre aussi plus d'informations sur les annotations sémantiques affectées par les changements effectués au cours de l'évolution.

D'autre part, durant le processus de l'évolution, nous avons montré la possibilité de propagation des changements de l'ontologie vers les parties dépendantes de l'ontologie modifiée. Cependant, nous n'avons pas beaucoup abordée a propagation des changements vers les ontologies dépendantes ou vers les applications reposant sur l'ontologie modifiée, mais nous nous sommes concentrés sur leur affectation sur les annotations sémantiques concernées. Comment détecter et corriger les annotations sémantiques qui deviennent inconsistantes après la modification des ontologies? Nous allons examiner ce problème dans le chapitre suivant.

Chapitre 5

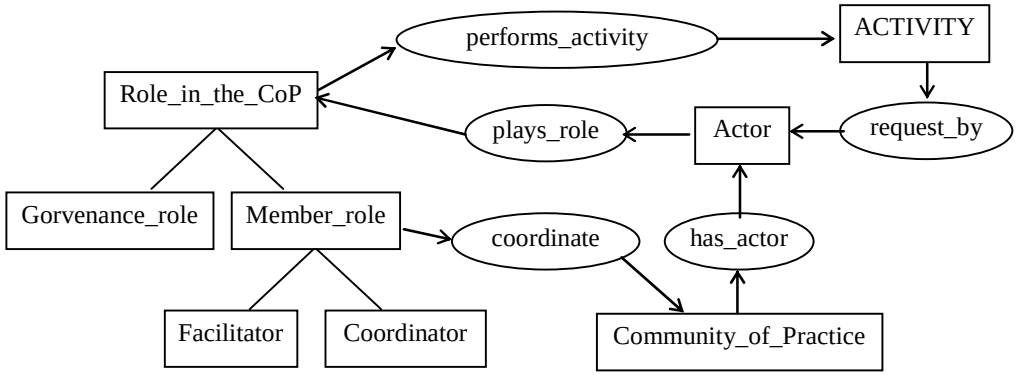
Évolution de l'annotation sémantique dans le contexte de modification de l'ontologie

Dans les chapitres précédents, nous avons souligné la nécessité pour les composants principaux du Web sémantique de pouvoir évoluer (i.e. ontologies et annotations sémantiques) afin de s'adapter à l'environnement distribué et dynamique en perpétuel changement. Nous avons présenté les aspects importants de l'évolution de l'ontologie et distingué également deux scénarios principaux du processus d'évolution de l'ontologie : (i) avec trace de changement et (ii) sans trace de changement. Dans ce chapitre, nous continuons à étudier les influences des changements de l'ontologie sur les annotations sémantiques. Et nous cherchons à répondre à la question : comment détecter et résoudre des annotations sémantiques inconsistantes générées à cause de modifications de l'ontologie? Pour ce faire, nous présentons *une approche procédurale* et *une approche basée sur des règles* qui sont implémentées dans le système CoSWEM en vue de gérer l'évolution des annotations sémantiques. Ces deux approches sont destinées aux deux scénarios d'évolution avec trace et sans trace de changement de l'ontologie respectivement.

5.1 Description du problème

5.1.1 Exemple de motivation

Nous avons présenté dans le chapitre 2 (c.f. section 2.2.2.2) les scénarios d'évolution de l'annotation sémantique et mis en exergue un scénario dans lequel les changements de l'ontologie de base peuvent affecter l'état consistant des annotations sémantiques concernées. Ce scénario semble être le plus fréquent dans la réalité dans le contexte où le fournisseur de l'ontologie fait des modifications sur son ontologie de base. Nous allons examiner dans cette section un exemple réel montrant les conséquences des modifications de l'ontologie développée dans le projet européen Palette [Vidou et al., 2007] sur la base d'annotations utilisant cette ontologie modifiée.



Extrait de l'ontologie O'CoP avant la modification

```
1.(r1 plays_role v1 .)
  (r1 type Actor .)(v1 type Role_in_the_CoP .)
2.(r2 performs_activity v2 .)
  (r2 type Role_in_the_CoP .) (v2 type ACTIVITY .)
3.(r3 performs_activity v3 .)
  (r3 type Member_role .) (v3 type ACTIVITY .)
4.(r4 coordinate v4 .)
  (r4 type Coordinator.)(v4 type Community_of_Practice.)
5.(r5 coordinate v5 .)
  (r5 type Facilitator.)(v5 type Community_of_Practice.)
6.(r6 request_by v6 .)
  (r6 type ACTIVITY .) (v6 type Actor .)
```

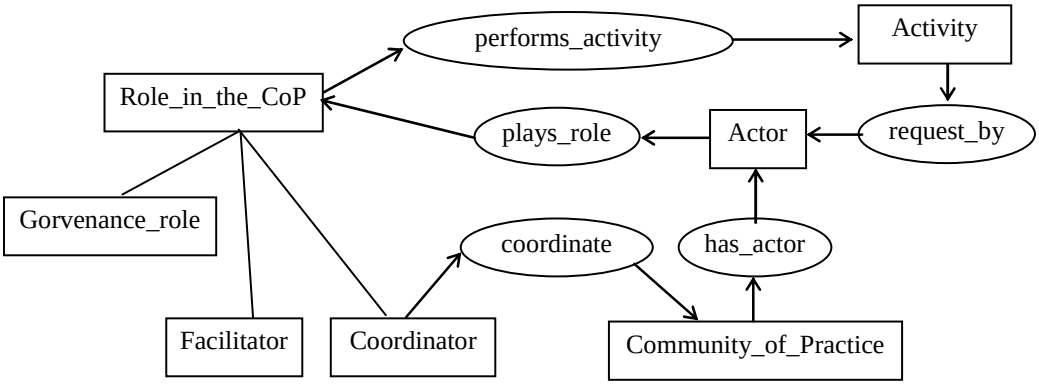
Triplets des annotations concernées par l'ontologie O'CoP

Figure 44 - Extrait de l'ontologie et les triplets d'annotations avant la modification

Revenons à l'exemple d'un extrait de l'ontologie O'CoP [Tifous et Dieng-Kuntz, 2007] qui est exploitée dans le projet Palette. La Figure 44 représente certains triplets d'annotations sémantiques qui reposent sur les termes définis de l'ontologie O'CoP. Nous avons des ressources annotées du type `Actor`, `Role_in_the_CoP`, `Member_role`, `ACTIVITY`, `Coordinator`, `Facilitator`, `Community_of_Practice`, etc. Ces ressources sont reliées par

des propriétés telles que *plays_role*, *performs_activity*, *coordinate*, etc.

Après avoir effectué certains changements sur l'ontologie O'CoP tels que la suppression du concept Member_role, ses deux sous-concepts sont reconnectés au concept Role_in_the_CoP, la propriété *coordinate* reçoit désormais le concept Coordinator comme domaine, le concept ACTIVITY est renommé en Activity, etc. Les triplets de l'annotation sémantique (c.f. Figure 44) ne seraient plus conformes à l'ontologie O'CoP à cause des changements ontologiques effectués, certains sont affectés par ces changements (c.f. lignes en gras 2, 3, 5 et 6, Figure 45) car ils contiennent des ressources qui font référence aux concepts modifiés par rapport à l'ancienne version. En plus, certains triplets deviennent maintenant inconsistants (c.f. lignes en gras et en rouge, Figure 45) du fait de la perte des liens qui font référence à des concepts correspondants dans l'ontologie avant sa modification.



Extrait de l'ontologie O'CoP après la modification

```
1.(r1 plays_role v1 .)
  (r1 type Actor .)(v1 type Role_in_the_CoP .)
2.(r2 performs_activity v2 .)
  (r2 type Role_in_the_CoP .) (v2 type ACTIVITY .)
3.(r3 performs_activity v3 .)
  (r3 type Member_role .) (v3 type ACTIVITY .)
4.(r4 coordinate v4 .)
  (r4 type Coordinator.)(v4 type Community_of_Practice.)
5.(r5 coordinate v5 .)
  (r5 type Facilitator.)(v5 type Community_of_Practice.)
6.(r6 request_by v6 .)
  (r6 type ACTIVITY .) (v6 type Actor .)
```

Triplets des annotations concernées par l'ontologie O'CoP

Figure 45 - Extrait de l'ontologie et les triplets d'annotations après la modification

À travers l'exemple ci-dessus, nous réaffirmons que non seulement des parties dépendantes de l'ontologie mais aussi certaines annotations sémantiques peuvent être affectées par les changements de l'ontologie. Les annotations sémantiques qui sont concernées par ces changements doivent être déterminées et gérées au cours de l'évolution afin d'assurer la consistance de toutes les

annotations sémantiques par rapport à l'ontologie de base. Pour atteindre ce but, une des tâches importantes est de formaliser le processus d'évolution de l'annotation sémantique du système CoSWEM. Dans la section suivante, nous définissons d'une manière formelle les terminologies et les étapes nécessaires au processus d'évolution de l'annotation sémantique.

5.1.2 Hypothèses

Dans le cadre de l'évolution d'un Web sémantique d'entreprise, nous considérerons l'ontologie comme une référence à respecter obligatoirement, en particulier par les annotations sémantiques. Si des changements sont effectués sur l'ontologie, les annotations devront s'adapter à la nouvelle version de l'ontologie et non l'inverse. Pour cette raison, notre travail sur l'évolution de l'annotation sémantique se base sur les hypothèses suivantes:

- Le travail consiste à gérer l'évolution de l'ontologie RDFS et la propagation des changements de l'ontologie vers les annotations RDF.

- Dans le cas de RDFS, nous considèrerons la sémantique du typage sous un aspect validation. Autrement dit, nous n'utiliserons pas l'inférence de type sur les annotations comme en RDF(S) classique, mais plutôt la propagation des changements de l'ontologie vers les annotations de manière à préserver leur compatibilité avec l'ontologie, celle-ci jouant toujours le rôle de référence. Dans notre travail, nous considérons qu'une "inconsistance" se produit s'il existe une utilisation des entités de l'annotation qui viole des contraintes de consistance définies auparavant.

Dans la section suivante, nous présentons nos définitions sur les contraintes de consistance, l'annotation sémantique inconsistante ainsi que les termes utilisés dans le processus d'évolution de l'annotation.

5.1.3 Définitions

Nous inspirons des recherches sur l'évolution de schémas dans les bases de données [Roddick, 1996] [Franconi et al., 2000], sur l'évolution de l'ontologie [Stojanovic, 2004] et sur la gestion de versions de l'ontologie [Klein et al., 2002], nous donnons notre vision de l'évolution de l'annotation sémantique ci-dessous :

Définition 1- Évolution de l'annotation sémantique

L'évolution de l'annotation sémantique est un processus d'adaptation d'une annotation sémantique donnée suite aux inconsistances générées à cause des changements sur l'ontologie de base ou sur l'annotation sémantique elle-même.

Nous avons mentionné auparavant les changements de l'ontologie affectant la consistance des annotations sémantiques concernées. En réalité, un annotateur peut modifier des entités d'une annotation sémantique sans référer aux liens des entités de l'ontologie de référence. Cela pourrait aussi entraîner une inconsistance sur cette annotation par rapport à son ontologie de référence. Cependant, le mécanisme de détection des inconsistances du système CoSWEM nous permet de détecter les inconsistances générées, à cause du changement ontologique mais aussi du changement sur l'annotation elle-même.

Afin de pouvoir décrire l'inconsistance d'une annotation sémantique, nous avons besoin tout d'abord de définir formellement le modèle d'annotation sémantique basé sur le modèle de données RDF présenté dans [Klyne et Carroll, 2004]. Ensuite, nous définirons la notion d'une contrainte de consistance et la notion d'une annotation sémantique inconsistante.

Définition 2 - Modèle d'annotation sémantique

Un modèle d'annotation est un tuple $M_A := (R_A, C_A, P_A, L, T_A)$*, où :*

– R_A *: ensemble des ressources*
– C_A *: ensemble des noms de concept définis dans l'ontologie* $(C_A \subset R_A)$
– P_A *: ensemble des noms de propriété définis dans l'ontologie* $(P_A \subset R_A)$
– L *: ensemble des valeurs littérales*
– T_A *: ensemble des triplets* $(s, p, v .)$ où $s \in R_A$, $p \in P_A$ et $v \in (R_A \cup L)$.

Selon le dictionnaire IEEE sur les glossaires standards d'ordinateur [IEEE, 1990], la consistance est le degré d'uniformité, de standardisation, et d'absence de contradiction entre les parties d'un système ou d'un composant. Au point de vue de la logique, la consistance est un attribut du système (logique) satisfaisant le fait qu'aucun des faits déductibles du modèle n'est contraire aux autres [Stojanovic, 2004]. Puisque nous nous intéressons à la consistance d'une annotation sémantique avant et après la modification de l'ontologie et de l'annotation sémantique, nous avons besoin de définir des contraintes de consistance qui doivent être réglées en vue d'assurer la consistance de tous les éléments de l'annotation sémantique.

Définition 3 – Contrainte de consistance

Une contrainte de consistance assure un accord cohérent de l'utilisation des entités de l'annotation sémantique en ce qui concerne son ontologie de référence.

Tous les changements affectant l'annotation sémantique doivent maintenir l'exactitude des contraintes de consistance. Lorsque nous faisons des changements sur l'ontologie, certaines opérations telles que le changement sur un label, sur un commentaire… n'introduisent aucune contradiction sur les annotations par rapport à leur ontologie de base. Autrement dit, elles n'affectent pas la consistance de l'annotation. Mais la plupart des opérations sur les concepts, les propriétés, les domaines/co-domaines ou les types de données vont impliquer des inconsistances

sur les annotations. Nous proposons donc ci-dessous des contraintes qui assurent l'utilisation d'une manière cohérente des termes (concepts, propriétés, domaine/co-domaine et type de données) définis dans le vocabulaire de l'ontologie pour annoter les ressources. Ces contraintes de consistance sont aussi définies pour la vérification de l'utilisation d'une manière cohérente des éléments de l'annotation sémantique par rapport son ontologie de base.

Afin d'exprimer des contraintes de consistance, nous prenons la notation présentée dans [Miller et Manola, 2004] pour décrire un triplet (`s p v.`) en RDF de l'annotation sémantique. Ce triplet constitue un énoncé sur la ressource et il peut être exprimé comme suit : le sujet `s` a une propriété *p* dont la valeur est `v`. Nous utilisons la primitive `rdf:type` pour indiquer qu'une ressource est de type ou de la classe spécifique (par exemple la ressource est de type `Concept` ou `Propriété`), et d'autres primitives avec le préfixe `rdfs:` pour décrire les classes ou les relations parmi ces classes dans l'ontologie.

- ***Contrainte sur le concept:*** Tous les types de concepts utilisés dans l'annotation doivent être définis auparavant dans l'ontologie.

  `(s rdf:type c)` $\Rightarrow$ `(c rdf:type rdfs:Class)`

- ***Contrainte sur la propriété:*** Toutes les types de propriétés utilisées dans l'annotation doivent être définies auparavant dans l'ontologie.

  `(s p v .)` $\Rightarrow$ `(p rdf:type rdf:Property)`

- ***Contrainte sur le domaine d'une propriété:*** Le type d'une ressource qui est le sujet d'une propriété dans l'annotation doit être compatible avec le domaine de la propriété correspondante définie dans l'ontologie.

  `(p rdf:type rdf:Property)`$\wedge$`(p rdfs:domain d)`$\wedge$ (d_1 `p v .)` $\Rightarrow$ (d_1 `rdf:type d)`

  `or` $\Rightarrow$ (d_1 `rdf:type d)`$\vee$ (d_1 `rdf:type dx) if` $\exists$`dx` $\in$ `descendants(d) such that (`d_1 `rdf:type dx)`

  (Note: `descendants(d)` est l'ensemble de concepts descendants de `d` dans la hiérarchie de concept.)

- ***Contrainte sur le co-domaine d'une propriété :*** Le type d'une ressource qui est la valeur d'une propriété dans l'annotation doit être compatible avec le co-domaine de la propriété correspondante définie dans l'ontologie.

  `(p rdf:type rdf:Property)`$\wedge$`(p rdfs:range r)`$\wedge$ `(s p` r_1 `.)`$\Rightarrow$ (r_1 `rdf:type r)`

$or \Rightarrow (r_1$ rdf:type $r) \vee (r_1$ rdf:type rx) if $\exists$rx $\in$ descendants(r) such that (r_1 rdf:type rx)

- ***Contrainte sur le type des données :*** Le type de données d'une valeur de propriété dans l'annotation doit être compatible avec la valeur de la propriété correspondante définie dans l'ontologie.

  $(p$ rdf:type rdf:Property$) \wedge (p$ rdfs:range $r) \wedge (r$ rdf:type rdfs:Datatype$) \wedge (s\ p\ r_1\ .) \Rightarrow (r_1$ rdf:type r)

Après l'exécution de chaque changement, les contraintes de consistance sont vérifiées pour savoir si l'état consistant de tout le système est encore garanti. Les annotations sémantiques deviennent inconsistantes si elles ne conservent pas l'exactitude de toutes les contraintes construites.

Définition 4 - Annotation sémantique inconsistante

Une annotation sémantique est définie comme étant inconsistante par rapport à son modèle d'ontologie si elle viole les contraintes de consistance définies pour le modèle d'annotation.

5.1.4 Les approches de l'évolution de l'annotation sémantique

La tâche de l'évolution de l'annotation sémantique est non seulement de détecter des annotations inconsistantes à cause des changements de l'ontologie mais aussi de les corriger en vue de garantir la consistance de toute la base d'annotations. En plus, elle a également pour le rôle d'informer les utilisateurs des changements dans la base d'annotations concernant les annotations inconsistantes.

Nous présentons dans cette section les différentes approches de l'évolution de l'annotation intégrées dans le système CoSWEM à l'aide d'un mécanisme de résolution des annotations inconsistantes qui utilise des stratégies de résolution. Nous les exposons en détail dans les sections suivantes.

Deux approches principales

Concernant les approches de résolution des inconsistances, [Stojanovic, 2004] a adopté les travaux existants dans la communauté des bases de données [Franconi et al., 2000] pour présenter leurs propositions sur les approches (i) approche procédurale et (ii) approche déclarative. Ces deux approches ont pour but de résoudre des problèmes de *sémantique du changement de l'ontologie* avec des *requêtes de changements spécifiées* par l'utilisateur. En revanche, notre proposition diffère entièrement du point de vue de l'objectif de deux approches proposées ; en effet, nous nous sommes concentrés sur la *résolution des*

*annotations sémantique*s *inconsistantes* et nous nous intéressons à *l'évolution sur deux contextes (i) avec trace et (ii) sans trace de changement de l'ontologie*.

- ***Approche procédurale :*** Elle vise à résoudre des inconsistances générées de l'annotation sémantique dans le cas où la trace de changement ontologique est conservée. Nous suivons alors des étapes séquentielles pour trouver des annotations inconsistances grâce aux informations enregistrées dans la trace d'évolution, et appliquons ensuite des procédures de résolution pour les corriger.

- ***Approche basée sur des règles :*** Dans l'environnement dynamique et distribué du Web sémantique, la création et la modification de l'ontologie peuvent être réalisées d'une manière collaborative, il n'est pas toujours possible de garder les historiques des changements de l'ontologie. Cette approche consiste à utiliser une base de règles de détection et de correction d'inconsistance pour chercher et réparer des annotations obsolètes et incohérentes par rapport à l'ontologie de base après sa modification.

Stratégies de résolution des annotations sémantiques

Comme nous l'avons abordé dans le chapitre précédent, un changement de l'ontologie peut être effectué selon différentes manières entraînant ainsi différents résultats. Puisque la taille et la complexité de l'ontologie grandissent, un changement sur un élément de l'ontologie pourrait affecter d'autres éléments dans la même ontologie. Il n'est pas envisageable qu'un ontologiste puisse bien connaître tous les composants d'une ontologie collaborative et bien contrôler tous les types de changements. Par conséquent, nous avons besoin de construire un ensemble de stratégies de résolution qui facilite, pour les ontologistes, la manipulation des opérations d'ontologies pour maximiser l'efficacité et l'exactitude de ces opérations [Luong et Dieng-Kuntz, 2007].

Dans le même ordre d'idée, nous construisons des stratégies de résolution pour les annotations sémantiques qui nous permettent de contrôler les inconsistances apparues sur les annotations. Au cours de l'évolution de l'ontologie, nous avons construit des stratégies de résolution pour chacun des changements ontologiques (section 4.3.2). En se basant sur ces stratégies, nous développons respectivement des stratégies pour chaque cas de résolution de l'annotation inconsistante. Le Tableau 8 représente un exemple de stratégies de résolution pour l'annotation sémantique équivalant à celles de l'ontologie pour un changement du type suppression d'un concept milieu dans l'ontologie. On supprime le concept milieu c ayant le super-concept c_2, les sous-concept c_i dans la hiérarchie des concepts et la propriété connectée p. Nous distinguons les différents cas de changements dans lesquels les sous-concepts c_i sont supprimés ou rebranchés au super-concept c_2 après la suppression de c. En effet, les résolutions pour les triplets contenant les ressources du type c et p sont les mêmes (c.f. Tableau 8)

dans ces cas, mais le traitement des triplets contenant les ressources du type c_i sont différents.

Au niveau de la structure de formalisation, le modèle de l'ontologie est plus complexe que celui de l'annotation sémantique. Cela demande donc une procédure de traitement des changements ontologiques plus compliquée avec différentes manières de résolution possibles. Bien qu'il existe plusieurs stratégies de résolutions pour l'ontologie, il y a seulement trois solutions principales pour corriger les annotations sémantiques inconsistantes. Autrement dit, nous avons trois types d'opérations ci-dessous pour corriger les triplets inconsistants de l'annotation.

- *Ne pas changer :* Les triplets sont conservés. On ne change rien sur les éléments de ces triplets.

- *Remplacer :* On peut remplacer un élément du triplet par un autre élément disponible et convenable. Par exemple, après avoir supprimé le concept `Member_role` (c.f. Figure 45), la ressource du type `Member_role` dans le triplet `(r3 type Member_role .)` (c.f. ligne 3, Figure 45) peut être remplacée par le type `Role_in_the_CoP`.

- *Supprimer :* On supprime tout le triplet contenant un élément inconsistant. Cette solution est appliquée seulement sur choix de l'utilisateur ou dans le cas où on ne peut pas trouver un autre élément qui pourrait remplacer celui inconsistant. Revenons à l'exemple illustré dans la Figure 45, les triplets `(r5 coordinate v5 .)` `(r5 type Facilitator .)` peuvent être supprimés car il n'existe plus le lien entre le concept `Facilitator` et la propriété *`coordinate`* dans l'ontologie après sa modification.

Tableau 8 - Stratégies de résolution pour l'ontologie et l'annotation sémantique

Stratégies de résolution pour l'ontologie		**Stratégies de résolution pour l'annotation sémantique**	
SO - 11	Traitement des instances du concept `c` - Supprimer les instances du concept `c`	**SA - 11**	Traitement des triplets contenant une instance du concept `c` **SA – 11a** Supprimer tous les triplets contenant une ressource du type `c`
	Attacher les instances du concept `c` au concept père c_2 de `c`		**SA – 11b** Remplacer le type d'une ressource de type `c` par le concept père c_2 de `c` , i.e. remplacer `(r type c)` par `(r type` c_2`)`

<table>
<tr><td rowspan="2">SO – 12</td><td rowspan="2"><u>Changement :</u> Supprimer le concept milieu c possédant un seul père c_2, c appartenant au domaine/co-domaine d'une propriété p. Les sous-concepts c_i de c sont aussi supprimés.
<u>Solution :</u>
- Supprimer le lien entre le concept c et son concept père c_2
- Supprimer le lien entre c et ses concepts fils c_i
- Supprimer le lien entre c et la propriété p
- Supprimer le concept c
- Supprimer les concepts fils c_i du concept c
Note : Traiter ensuite les concepts fils c_i du concept c</td><td rowspan="3">SA – 12</td><td>SA – 12a
Si aucun ancêtre de c n'appartient au domaine (respectivement co-domaine) de la propriété p, alors supprimer tous les triplets contenant à la fois une ressource du type c et la propriété p , i.e. les triplets (r p v) avec (r type c) ou (respectivement les triplets (r1 p r) avec (r type c))</td></tr>
<tr><td>SA – 12b
S'il existe un ancêtre du concept c appartenant au domaine (respectivement co-domaine) de la propriété p, alors remplacer le type c d'une ressource de type c apparaissant dans un triplet avec la propriété p par le concept père c_2 , i.e. remplacer (r type c) par (r type c_2) s'il existe un triplet (r p v) ou (respectivement (r1 p r))</td></tr>
<tr><td>Autre option : Remplacer le type c d'une ressource de type c apparaissant dans un triplet avec la propriété p par le concept c_x indiqué par l'utilisateur.
Note : Traiter ensuite les triplets contenant à la fois la propriété p et une ressource de type c_i</td></tr>
<tr><td rowspan="2">SO – 13</td><td><u>Changement :</u> Supprimer le concept milieu c possédant un seul père c_2, c appartenant au domaine/co-domaine d'une propriété p. Les sous-concepts c_i de c sont rebranchés au super-concept c_2
<u>Solution :</u>
- Supprimer le lien entre le concept c et son concept père c_2</td><td rowspan="2">SA – 13</td><td>SA – 13a
Si aucun ancêtre de c n'appartient au domaine (respectivement co-domaine) de la propriété p, alors supprimer tous les triplets contenant à la fois une ressource du type c et la propriété p , i.e. les triplets (r p v) avec (r type c) ou (respectivement les triplets (r1 p r) avec (r type c))</td></tr>
<tr><td>- Supprimer le lien entre c et ses concepts fils c_i
- Supprimer le lien entre le c et la propriété p
- Supprimer le concept c
- Rebrancher les concepts fils c_i du concept c au concept père c_2</td><td>SA – 13b
S'il existe un ancêtre du concept c appartenant au domaine (respectivement co-domaine) de la propriété p, alors remplacer le type c d'une ressource de type c apparaissant dans un triplet avec la propriété p par le concept père c_2 , i.e. remplacer (r type c) par (r type c_2) s'il existe un triplet (r p v) ou (respectivement (r1 p r))</td></tr>
</table>

			Autre option : Remplacer le type c *d'une ressource de type* c *apparaissant dans un triplet avec* p *par le type* c_x *indiqué par l'utilisateur.* *Note : Garder les triplets contenant à la fois* p *et une ressource de type* c_i *, s'il existe un lien entre un ancêtre du concept* c *et* p*. Sinon, supprimer ces triplets contenant* p *et une ressource de type* c_i

Résolution des annotations sémantiques inconsistantes

Il y a deux manières principales de résoudre des inconsistances d'annotations sémantiques :

- ***Automatique :*** Une annotation inconsistante peut être corrigée automatiquement selon les stratégies de résolution définies auparavant. Ce genre de résolution est convenable pour le type d'évolution où on peut garder la trace de changements. Puisqu'on connaît les changements ontologiques effectués ainsi que les stratégies utilisées pour réparer l'inconsistance de l'ontologie, on peut alors appliquer les stratégies correspondantes pour les annotations. Une stratégie par défaut permet à l'utilisateur de réaliser le processus selon des solutions déterminées, mais ce n'est pas toujours la meilleure pour tout le monde, on a besoin de plus d'options pour traiter les inconsistances selon ses préférences.

- ***Semi-automatique :*** Idéalement il serait plus concluant qu'un utilisateur puisse corriger chaque inconsistance selon sa préférence. CoSWEM permet à l'utilisateur de corriger chaque type d'inconsistance ainsi que chaque annotation inconsistante. Il fournit aussi des suggestions de modification : par exemple si on veut remplacer un concept supprimé par un autre concept de l'ontologie, CoSWEM propose une liste de concepts disponibles de la nouvelle ontologie en représentant les concepts les plus pertinents à remplacer au début de liste. Dans ce cas, un algorithme a été implémenté dans CoSWEM pour chercher les concepts de la nouvelle ontologie qui sont proches avec le concept supprimé, ensuite le "meilleur" concept (i.e. le plus proche) est choisi à remplacer celui obsolète dans l'annotation.

5.2 Approche procédurale - évolution de l'annotation sémantique avec trace de changements

5.2.1 Processus d'évolution avec la trace de changement

Le processus d'évolution appliqué dans l'approche procédurale peut être divisé en trois étapes principales comme suit (c.f. Figure 46) :

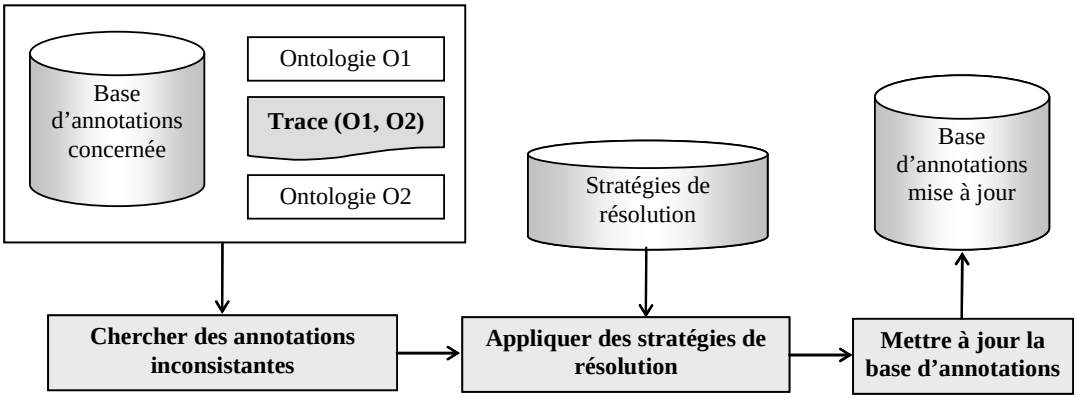


Figure 46 - Processus d'évolution pour l'approche procédurale

- *Chercher des annotations inconsistantes :* Les annotations inconsistantes sont trouvées grâce aux informations sur les changements effectués capturés dans la trace de changement. Cette étape nous renvoie une liste d'annotations sémantiques qui ont besoin d'être corrigées à cause de leur inconsistances internes par rapport à leur ontologie de base.

- *Appliquer des stratégies de résolution :* Nous appliquons des stratégies de résolution construites pour les annotations sémantiques inconsistantes. Cette phase peut être aussi réalisée selon la préférence de l'utilisateur pour corriger les inconsistances selon son choix.

- *Mettre à jour la base d'annotations :* Après avoir vérifié et corrigé les inconsistances générées, la base d'annotations est mise à jour avec les annotations corrigées consistantes. Cette phase garantit la cohérence de la base d'annotations pour de futurs usages.

5.2.2 Approche procédurale

Recherche des annotations sémantiques inconsistantes

Nous examinons l'exemple présenté dans la section 5.1 (c.f. Figure 44, Figure 45) sur la modification de l'ontologie. En effet, l'ontologiste a supprimé le concept `Member_role` et relié le concept `Coordinator` à la propriété *`coordinate`* (c.f. Figure 47). Afin d'assurer la consistance de toute l'ontologie après avoir supprimé le concept `Member_role`, nous devons également traiter les sous-concepts, les instances du concept supprimé ainsi que les liens avec d'autres propriétés. Un ensemble de changements additionnels sont donc générés pour traiter ces éléments concernés. La Figure 47 présente les changements additionnels à effectuer pour compléter l'opération de suppression du concept `Member_role` : (i) relier le concept `Facilitator` au super-concept `Role_in_the_CoP`, (ii) relier le concept `Coordinator` au super-concept `Role_in_the_CoP` et (iii) et enlever le lien entre le concept `Member_role` et la propriété *`coordinate`*.

Dans le chapitre 4, nous avons mentionné la possibilité de capturer et de représenter les changements effectués dans une trace d'évolution grâce à un éditeur d'ontologie (i.e. ECCO). Cette trace d'évolution enregistrant les changements effectués ci-dessus est représentée sous forme d'annotation sémantique qui nous facilite d'exploitation des informations sur la modification de l'ontologie. Dans le but d'illustrer l'exploitation de cette trace d'évolution, elle est représentée d'une manière plus simple comme la Figure 48. Dans cette figure, à part des concepts ou des propriétés de l'ontologie d'évolution décrivant les informations du processus d'évolution (e.g. `TraceOnto`, `DeleteConcept`, `hasChange`, `hasAdditionalChange`, etc.), les entités modifiées de l'ontologie de domaine (i.e. l'ontologie O'CoP) sont également enregistrées comme les concepts `Member_role`, `Facilitator`, `Coordinator`, `Role_in_the_CoP` et la propriété *`coordinate`*.

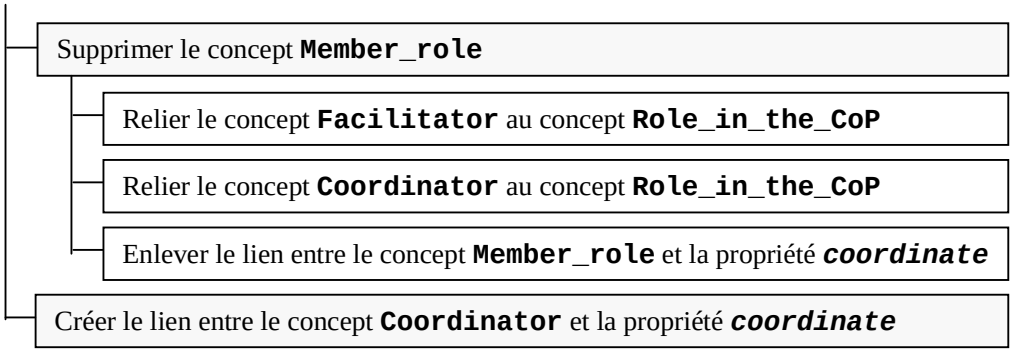


Figure 47 - Changements de l'ontologie effectués

Après une étape intermédiaire d'extraction des éléments modifiés de l'ontologie de domaine, nous obtenons une liste des concepts et des propriétés concernés par les changements de l'ontologie. Grâce à cette liste, nous pouvons

chercher les annotations sémantiques devenant inconsistances à cause de la perte des liens de références avec l'ontologie de base.

```
<TraceOnto>
   <hasChange>
      <DeleteConcept >
         <hasDeleteConcept Member_role>
         <hasAnnotInconsistency>yes</hasAnnotInconsistency>
            <hasAdditionalChange>
               <CreateHierarchyConceptLink>
                <hasAnnotInconsistency>no</hasAnnotInconsistency>
                 <hasSubConceptCreate Facilitator >
                 <hasSuperConceptCreate Role_in_the_CoP >
               <CreateHierarchyConceptLink>
            </hasAdditionalChange>
            <hasAdditionalChange>
               <CreateHierarchyConceptLink>
                 <hasAnnotInconsistency>no</hasAnnotInconsistency>
                 <hasSubConceptCreate Coordinator >
                 <hasSuperConceptCreate Role_in_the_CoP  >
               <CreateHierarchyConceptLink>
            </hasAdditionalChange>
            <hasAdditionalChange>
               <RemoveConceptDomainLink>
                   <hasAnnotInconsistency>yes</hasAnnotInconsistency>
                   <hasConceptDomainCreate Member_role >
                   <hasPropertyDomainCreate coordinate >
                <RemoveConceptDomainLink>
            </hasAdditionalChange>
      </DeleteConcept>
    </hasChange>

    <hasChange>
      <CreateConceptDomainLink>
          <hasAnnotInconsistency>no</hasAnnotInconsistency>
          <hasConceptDomainCreate Coordinator >
          <hasPropertyDomainCreate coordinate >
      <CreateConceptDomainLink>
    </hasChange>
<TraceOnto>
```

Figure 48 – Exemple d'une trace d'évolution

Nous avons mentionné dans le chapitre précédent les annotations potentiellement inconsistantes qui sont concernées par les changements de l'ontologie effectués. Ces annotations potentiellement inconsistantes peuvent inclure des annotations consistantes et d'autres vraiment inconsistantes qui contiennent des triplets inconsistants. Nous pouvons distinguer ces types d'annotation grâce à un mécanisme de vérification des changements de correction obligatoire et facultative. Par exemple, nous pouvons déterminer les changements de correction obligatoire (i.e. `DeleteConcept`,

RemoveConceptDomainLink) dans la trace d'évolution (c.f. Figure 48) par la valeur (i.e. Yes) de la propriété <hasAnnotInconsistency>. L'extraction des changements entraînant la correction obligatoire (ou facultative), des éléments et des annotations inconsistantes… peut être réalisée d'une manière automatique par la fonction d'inférence du système CoSWEM. Cette fonction utilise la bibliothèque API de Corese qui est développée pour réaliser la recherche sémantique des entités de l'ontologie et de la base d'annotations.

Dans cette étape, CoSWEM utilise le moteur de recherche sémantique Corese pour lancer les différentes requêtes sur la base d'annotations en vue de chercher les annotations concernées. Ces requêtes sont formulées en syntaxe SPARQL[39] [Prud et Seaborne, 2007] qui est le langage de requête pour RDF recommandé par le W3C. La Figure 49 présente un exemple d'exécution d'une requête en SPARQL par Corese. Nous avons besoin de déterminer les triplets de l'annotation contenant le concept supprimé Member_role.

```
PREFIX oc: <http://www.inria.fr/acacia/ontocreator#>
select * where {
GRAPH ?s {
   ?suj ?prop ?obj
   ?suj rdf:type ?type
FILTER (?type ~ "Member_role")
}
```

Figure 49 - Exemple d'une requête envoyée à Corese en langage SPARQL

Le résultat de cette requête nous renvoie des triplets dont la ressource est de type Member_role. Par exemple, l'interface graphique de Corese (c.f. Figure 50) représente un triplet trouvé contenant le sujet ?suj (i.e. #WI-Memrole1) , la propriété ?prop (i.e. performs_activity) et l'objet ?obj (i.e. #WI-AC2). Le type de la ressource sujet est le concept #Member_role. Le mécanisme de recherche par le moteur Corese est intégré dans le système CoSWEM qui nous permet d'obtenir le même résultat.

Avec cette requête, nous pouvons aussi savoir quelles sont les annotations sémantiques qui contiennent les triplets trouvés. Plus précisément dans cet exemple, nous récupérons l'emplacement physique du fichier d'annotation sémantique annot4-WI.rdf (i.e. *C:/Program Files/Apache Software Foundation/Tomcat 5.5/webapps/coswem-ifi/WEB-INF/coswem-withouttrace/data/annotations/annot4-WI.rdf*) qui contient les triplets concernant le concept supprimé Member_role (c.f. Figure 50).

Un autre exemple est illustré dans la Figure 51, les annotations inconsistantes détectées (i.e. fichiers annot2.rdf et annot5.rdf) concernent le concept supprimé personnes_physique3. Nous pouvons sélectionner une ou des annotations inconsistantes à corriger selon la préférence de l'utilisateur.

[39] http://www.w3.org/TR/rdf-sparql-query/

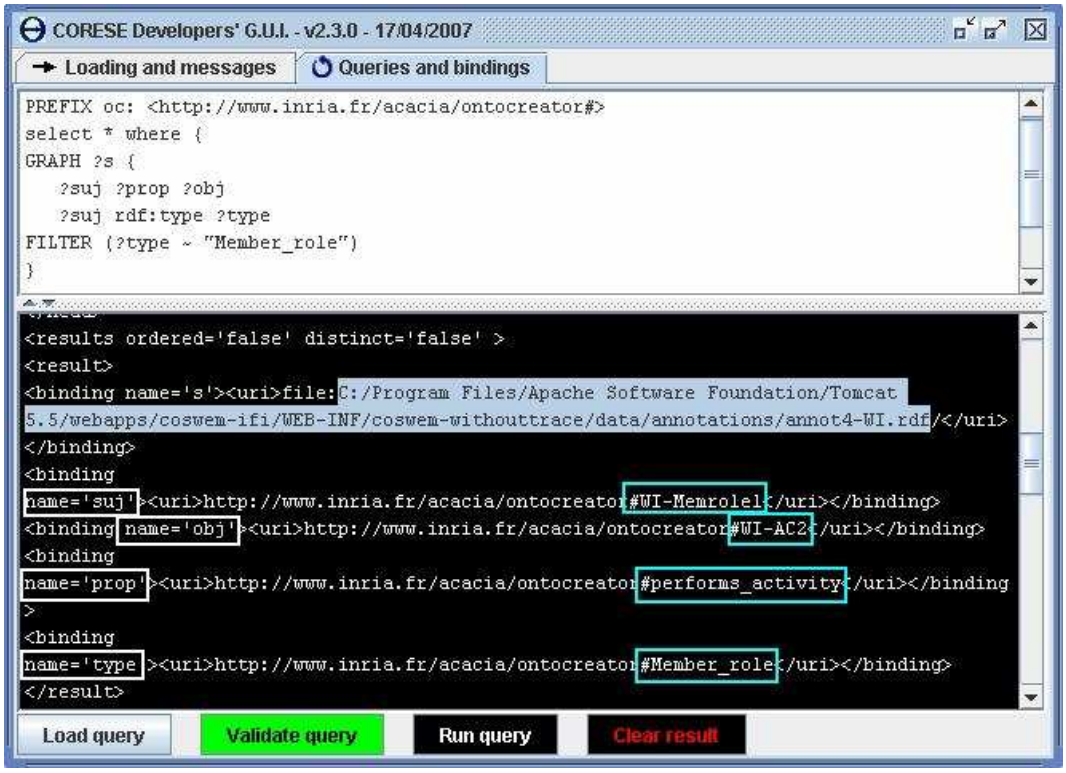


Figure 50 - Résultat d'une requête de recherche effectuée par Corese

Application des stratégies de résolution

Si nous n'effectuons que des changements sur l'ontologie mais ne faisons pas attention à la base d'annotations reposant sur cette ontologie, alors certaines annotations inconsistantes peuvent causer des problèmes en utilisant des termes ontologiques d'une manière incohérente. Par conséquent, il nous faut également traiter toutes les annotations inconsistantes détectées.

Nous obtenons à cette étape une liste d'annotations sémantiques où manquent un ou plusieurs liens de référence vers leur ontologie de base. Pour chaque annotation inconsistante détectée, nous connaissons également les triplets affectés au sein de cette annotation. Dans ce cas, le système peut les corriger d'une manière *automatique*. En analysant la trace d'évolution, il sait comment les ontologies ont évolué. Autrement dit, il eut savoir quels changements ont été effectués et dans quel ordre d'exécution, quelles entités de l'ontologie sont affectées par ces changements ontologiques, quelles stratégies de résolution ont été appliquées pour résoudre quel changement, etc. Il sait la stratégie de résolution qui est appliquée pour l'ontologie. En nous basant sur ces informations, les stratégies de résolution appropriées correspondantes seront choisies pour appliquer aux annotations sémantiques inconsistantes détectées.

Cependant, le processus de correction d'inconsistance peut être réalisé d'une manière semi-automatique grâce à l'intervention de l'utilisateur. L'utilisateur (e.g. un ontologiste ou un annotateur) attend toujours plus d'options dans le cas où il veut réparer des inconsistances selon sa préférence au moment de modifier ces annotations inconsistantes. Le système CoSWEM fournit des interfaces qui facilitent à l'utilisateur le choix d'une opération appropriée. Par exemple, nous pouvons choisir de remplacer ou de supprimer les triplets des annotations inconsistantes (i.e. fichiers annot2.rdf et annot5.rdf) contenant le concept supprimé `personnes_physique3`. Dans l'interface de résolution du concept inconsistant `personnes_physique3` (i.e. Figure 51), une liste des concepts disponibles de la nouvelle ontologie est affiché : les concepts similaires du concept supprimé sont classés au début de la liste, les autres moins similaires sont affichés dans le menu *drop-down* (i.e. le concept `Governance_role` et les concepts restants). L'utilisateur va choisir un concept à remplacer selon sa préférence ou il peut chosir l'option de supprimer les triplets contenant ce concept inconsistant `personnes_physique3.`

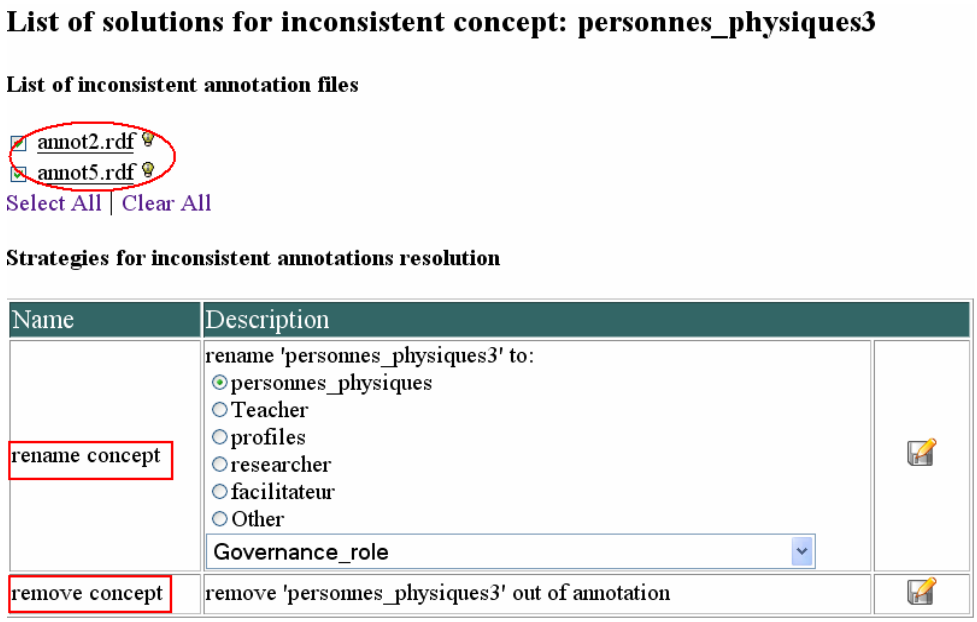


Figure 51 - Annotations inconsistantes trouvées concernant le concept supprimé `personnes_physique3`

Mise à jour de la base d'annotations sémantique

Cette phase a pour but de mettre à jour toutes les annotations corrigées de la base d'annotations. Dans notre système CoSWEM, nous avons la possibilité de garder plusieurs versions différentes des annotations sémantiques pendant le processus d'évolution. Lors de la correction des annotations inconsistantes, nous pouvons choisir de les remplacer par les nouvelles versions correspondantes ou de garder les versions existantes et enregistrer parallèlement les annotations corrigées en nouvelles versions. Cela nous laisse la possibilité d'utiliser la version que l'on

veut au cours de l'évolution dans un système de gestion des versions. Nous pouvons choisir de travailler avec une version quelconque de l'annotation selon notre configuration dans CoSWEM.

5.3 Approche basée sur des règles - évolution de l'annotation sémantique sans trace de changements

5.3.1 Processus d'évolution sans trace de changement

Il arrive souvent que l'évolution de l'ontologie ne permette pas toujours de garder les traces entre deux versions différentes de l'ontologie. La raison principale est que les ontologies ont tendance être développées de manière collaborative et dans le contexte dynamique et distribué du Web sémantique. Normalement, on effectue des changements sur une version de l'ontologie et on garde finalement la dernière (ou la plus récente) version.

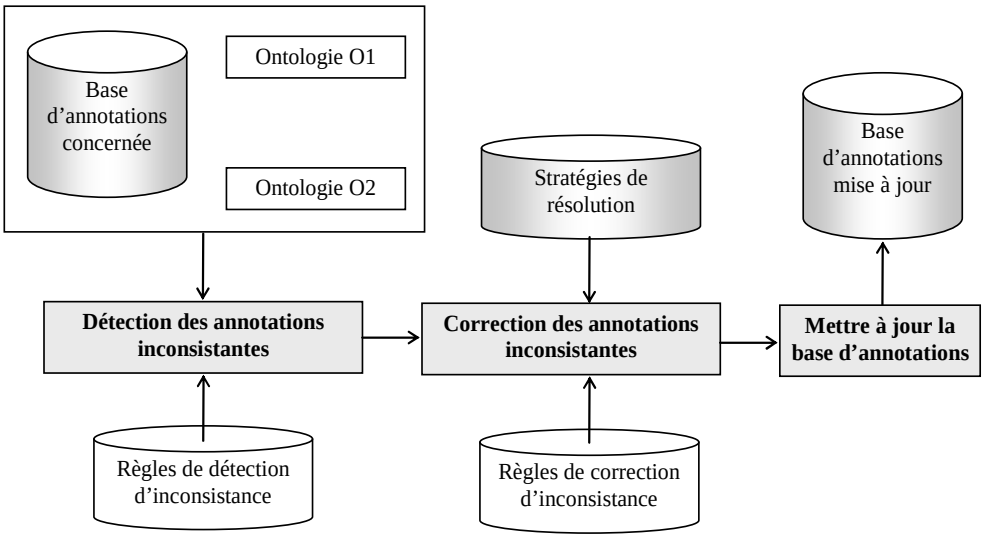


Figure 52 - Processus d'évolution pour l'approche basée sur des règles

L'approche que nous proposons dans ce contexte est basée sur les règles de détection et de correction d'inconsistance. Elles sont construites à partir des contraintes de consistance qui doivent être satisfaites pour le modèle d'annotation. Nous avons défini des contraintes de consistance comme pouvant être considérées comme un accord cohérent pour les entités d'annotations sémantiques en ce qui concerne leur ontologie de base (c.f. section 5.1). En reposant sur ces contraintes, nous développons une approche basée sur des règles dans un processus comme ci-dessous (c.f. Figure 52):

- *Détecter des annotations sémantiques inconsistantes :* Cette phase consiste à tester l'ensemble des règles de détection d'inconsistance qui

sont dédiées à certains aspects tels que le concept, la propriété, le domaine/co-domaine et le type des données. Le système CoSWEM vérifiera si ces règles satisfont les contraintes de consistances définies ou pas. Cette phase est réalisée automatiquement et renvoie les annotations inconsistantes par rapport aux changements de l'ontologie effectués.

- *Corriger des annotations sémantiques inconsistantes :* Après avoir déterminé les annotations inconsistantes réelles, elles seront corrigées en appliquant les règles de correction d'inconsistance. Nous avons établi les stratégies possibles qui résolvent la propagation des changements ontologiques vers leurs annotations sémantiques afin de conserver l'état consistant de ces annotations. L'utilisateur peut aussi choisir l'opération à appliquer selon leurs besoins.

- *Mettre à jour la base d'annotations :* La base d'annotations est mise à jour par les annotations corrigées consistantes dans cette phase. Elle assure l'utilisation cohérente de la base d'annotations pour les phases suivantes.

5.3.2 Détection des inconsistances de l'annotation sémantique

Définition et algorithme des règles de détection

Cette phase de détection d'inconsistance nous permet de trouver les annotations inconsistantes qui deviennent obsolètes par rapport à l'ontologie modifiée. Elle est réalisée grâce à un ensemble de règles de détection d'inconsistances qui sont basées sur les contraintes de consistance.

Soit C_O et P_O les ensembles des concepts et des propriétés de l'ontologie `O`, soit C_A et P_A les ensembles des concepts et des propriétés figurant dans l'annotation `A`. Soit D_O et D_A les ensembles de tous les types de données définis respectivement dans l'ontologie `O` et dans l'annotation `A`. Tous les triplets dans l'annotation ont la forme `(r` p_t `v .)` avec la ressource `r` du type c_t `(r type` c_t `.)`. Nous utilisons aussi la fonction `domain(p)` (respectivement `range(p))` pour indiquer les concepts appartiennent au domaine (respectivement co-domaine) de la propriété *p*. La fonction `ancestor(c)` rend un ensemble des concepts ascendants de `c` dans l'ontologie et la fonction `datatype(v)` nous donne le type de données d'une valeur `v`. Nous avons construit des groupes de règles qui peuvent détecter les inconsistances concernant les concepts, les propriétés, les domaine/co-domaine, les types des données (datatype). Dans le paragraphe suivant, nous présentons certaines règles qui peuvent être utilisées pour illustrer l'exemple mentionné précédemment.

Groupe 1 (règles de détection pour la ressource de concept) : Si un type de concept est utilisé dans l'annotation mais il n'est pas défini dans l'ontologie de base, alors l'annotation mène à un état inconsistant.

R-1 :∀ annotation A, Si ($c_t \in C_A$) et ($c_t \notin C_0$) Alors marquer(inconsistant)

R-2 :∀ annotation A, Si ($c_t \in C_A$) et ($c_t \in C_0$) Alors marquer (OK)

Groupe 2 (règles de détection pour la ressource de propriété) : Si une propriété est utilisée dans l'annotation mais qu'elle n'est pas encore définie dans l'ontologie, alors l'annotation mène à un état inconsistant.

R-3 : ∀ annotation A, Si ($p_t \in P_A$) et ($p_t \notin P_0$) Alors marquer(inconsistant)

R-4 : ∀ annotation A, Si ($p_t \in P_A$) et ($p_t \in P_0$) Alors marquer(OK)

Groupe 3 (règles de détection pour la ressource qui est le domaine de propriété) : Si une propriété p_t prend une valeur qui est une ressource r du type de concept c_t dans une annotation sous la forme (r p_t v .) (r type c_t .) (v type c_v.), mais ni c_t ni aucun de ses ancêtres n'appartiennent au domaine de p_t dans l'ontologie, alors cette annotation mène à un état inconsistant.

-R-5 : ∀ annotation A, Si ($c_t \notin$ domain(p_t)) et ∀c ∈ ancestors(c_t), c ∉ domain(p_t) Alors marquer(inconsistant)

-R-6 : ∀ annotation A, If ($c_t \in$ domain(p_t)) ou ∃c ∈ ancestors(c_t), c ∈ domain (p_t) Alors marquer(OK)

Groupe 4 (règles de détection pour la ressource qui est le co-domaine de propriété) : Si une propriété p_t prend une valeur qui est une ressource v du type de concept c_v dans une annotation sous la forme (r p_t v .) (r type c_t .) (v type c_v.), mais ni c_v ni aucun de ses ancêtres n'appartiennent au co-domaine de p_t dans l'ontologie, alors cette annotation mène à un état inconsistant.

-R-7 : ∀ annotation A, Si ($c_v \notin$ range(p_t)) et ∀c ∈ ancestors(c_v), c ∉ range(p_t) Alors marquer(inconsistant)

-R-8 : ∀ annotation A, If ($c_v \in$ range(p_t)) ou ∃c ∈ ancestors(c_v), c ∈ range (p_t) Alors marquer(OK)

Groupe 5 (règles de détetion pour le type de données): Si une valeur v de la propriété p_t a un type de données d_t dans l'annotation, mais d_t n'a pas été défini comme le type de données de la valeur de p_t dans l'ontologie, alors cette annotation mène à un état inconsistant.

-R-9 : ∀ annotation A, Si (datatype(v)= d_t) et ($d_t \in D_A$) et ($d_t \notin D_0$) Alors marquer(inconsistant)

-R-10 : ∀ annotation A, Si (datatype(v)= d_t) et ($d_t \in D_A$) et ($d_t \in D_O$) Alors marquer(OK)

Pour cette phase, nous avons implémenté des algorithmes qui nous permettent de vérifier des règles proposées et de détecter les annotations inconsistantes. Ces algorithmes vont comparer les éléments entre la nouvelle version de l'ontologie et la base d'annotations concernée pour trouver les différences qui serviront au processus de détection des inconsistances. La Figure 53 représente un algorithme permettant de chercher tous les types de concept utilisés dans les annotations qui deviennent inconsistantes par rapport à la nouvelle version de l'ontologie. Cet exemple concerne aussi les règles de détection des inconsistances générées sur les ressources de type concept.

```
1     Algorithm.InconsistentConceptDetection(O,A)
2     Input: ontology O, set of annotations related to
3       ontology O
4     Output: list of inconsistent concepts IC_A
5     Begin
6           Load O and A
7           /*get concept name of related annotations*/
8           C_A ← getConceptName(A)
9           For Each name in C_A Do
10          /*validate obsolete concepts by Corese. This
11          Boolean function returns a true value if
12          name∈O*/
13          result := validateInconsistentConcept(name, O)
14                If result=true Then
15                      NextFor
16                Else
17                      Message('inconsistent');
18                      name → IC_A
19                EndIf
20          EndFor
21    End
```

Figure 53 - Algorithme de détection des inconsistances sur les concepts

```
1     <oc:Member_role rdf:about="http://www.inria.fr/acacia/ontocreator#WI-
2     Memrole1">
3           <oc:performs_activity
4           rdf:resource="http://www.inria.fr/acacia/ontocreator#WI-AC2"/>
5     </oc:Member_role>
6
7     <oc:Coordinator rdf:about="http://www.inria.fr/acacia/ontocreator#WI-
8     Coor1">
9           <oc:coordinate
10          rdf:resource="http://www.inria.fr/acacia/ontocreator#WI-CoP1"/>
11    </oc:Coordinator>
12
13    <oc:Facilitator rdf:about="http://www.inria.fr/acacia/ontocreator#WI-
14    Faci1">
15          <oc:coordinate
16          rdf:resource="http://www.inria.fr/acacia/ontocreator#WI-CoP1"/>
17    </oc:Facilitator>
18
19    <oc:ACTIVITY rdf:about="http://www.inria.fr/acacia/ontocreator#WI-
20    AC_tmp1">
21          <oc:order_by
22          rdf:resource="http://www.inria.fr/acacia/ontocreator#WI-ACT_1"/>
23    </oc:ACTIVITY>
```

Figure 54 - Exemple des triplets inconsistants détectés

Nous appliquons les règles et les algorithmes de détection construits à l'ensemble d'annotations qui repose sur l'ontologie modifiée. Revenons à l'exemple précédent (c.f. Figure 45), en appliquant la règle `R-1`, nous pouvons détecter les triplets dans les lignes 2, 3 et 6 (c.f. lignes en gras et en rouge, Figure 45) qui deviennent inconsistants à cause de la perte de référence des ressources du triplet vers les concepts `Member_role` et `ACTIVITY`. La règle `R-5` détecte les triplets dans la ligne 5 (c.f. lignes en gras et en rouge, Figure 45) inconsistants parce que la relation de domaine entre le concept `Facilitator` et la propriété *`coordinate`* ont été enlevées dans la nouvelle version de l'ontologie. La Figure 54 représente certains triplets détectés dans la base d'annotations qui sont inconsistants par rapport aux changements de l'ontologie effectués. A partir de ces triplets, nous pouvons retrouver les annotations inconsistantes qui les contiennent.

Représentation des règles de détection d'inconsistance

Les règles de détection proposées dans la section ci-dessus sont représentées à l'aide d'un fichier XML dans le système CoSWEM. Cette représentation facilite la formulation et la modification des règles. Nous pouvons ajouter une nouvelle règle, modifier une règle existante ou enlever une règle obsolète sans affecter le fonctionnement du système.

```
1       <rules>
2          <rule id="R-1"
                handler="fr.inria.acacia.coswem.model.impl.-
                ConceptRuleHandler"
3               name="check for the existence of concept">
4               <query type="concept">
5               <. . .>
6               <queryString>
7                       ${PREFIX_TEMPLATE}
8                       select * ${MAX_RESULT} where {
9                       source ?src {
10                              ?suj ?prop ?obj .
11                              ?suj rdf:type ?type .
12                              FILTER ( ?type ~ '#${TOKENCLASS}')}
13                      }
14              </queryString>
15              </query>
16              <param key="INCONSISTENY_XSL" value="WEB-
17              INF/coswem-withouttrace/xsl/concept.xsl"/>
18              <. . .>
19      </rule>
```

Figure 55 - Représentation d'une règle de détection d'inconsistance

Dans ces règles, nous intégrons des requêtes en SPARQL pour chercher des triplets qui remplissent une condition quelconque, par exemple : rechercher des triplets contenant des ressources du type d'un concept donné par l'utilisateur (c.f. Figure 55). Ces requêtes nous renvoient aussi leur emplacement physique (i.e. le chemin du fichier d'annotation). Pour la phase de détection des inconsistances, ces requêtes sont soumises au moteur de recherche sémantique Corese, qui va les projeter sur la base d'annotations pour trouver des réponses à ces requêtes. D'autre part, nous pouvons également paramétrer ces règles pour les traitements

techniques dans le système CoSWEM (e.g. le nombre de résultats renvoyés, le paramètre du filtrage, la feuille de présentation XSLT, etc.)

5.3.3 Correction des inconsistances de l'annotation sémantique

Règles de correction d'inconsistance

Après avoir détecté toutes les annotations sémantiques inconsistantes dans la base d'annotations par rapport aux changements de l'ontologie, nous devons corriger ces inconsistances en appliquant les règles de correction sur ces annotations. Puisque la trace d'évolution n'est pas gardée, nous ne savons pas comment l'ontologie a évolué. Dans ce cas, en vue d'appliquer *automatiquement* la phase de correction, nous utilisons des règles de correction qui guident la résolution des annotations inconsistantes afin de conserver une consistance globale.

Pour modéliser ces règles, nous avons besoin de plus d'information sur les éléments de l'annotation inconsistante et aussi sur ceux des deux versions de l'ontologie : l'ancienne version O_1 (avant la modification) et la nouvelle version O_2 (après la modification). Nous utilisons des notations CO_1, CO_2, C_A pour indiquer respectivement les ensembles de concepts de l'ontologie O_1, O_2 et de l'annotation A. Les notations PO_1, PO_2, P_A représentent respectivement les ensembles de propriétés de l'ontologie O_1, O_2 et l'annotation A. Les notations HC_{01}, HC_{02} représentent les hiérarchies des concepts de O_1 et O_2, par exemple $(c, c_2) \in HC_{01}$ décrit une relation entre le concept c et son père c_2 dans l'ontologie O_1. D'autre part, si un triplet est inconsistant, il contient alors au moins un élément inconsistant, par exemple $(c_{ins}\ p\ v\ .)$ ou $(c\ p_{ins}\ v\ .)$ ou $(c_{ins}\ p_{ins}\ v\ .)$, etc. Selon les contraintes de consistance définies avant (c.f. section 5.1), un élément est inconsistant (i.e. c_{ins}, p_{ins}) s'il existe dans l'ancienne version de l'ontologie O_1 et de l'annotation A mais n'existe pas dans la nouvelle version d'ontologie O_2. Basées sur ces notations, nous construisons certaines règles de correction. La section suivante présente les exemples de règles de correction qui sont appliquées sur les triplets $(c_{ins}\ p\ v\ .)$ contenant l'élément inconsistant du type concept c_{ins}.

- RC-1 : S'il n'existe pas un concept père c_2 qui est le père de c_{ins} dans les deux versions O_1 et O_2, Alors supprimer le triplet inconsistant (e.g. supprimer $(c_{ins}\ p\ v\ .)$).

  Si $\neg\exists c_2 : c_2 \in O_1, c_2 \in O_2, (c_{ins}, c_2) \in HC_{01}$ Alors supprimer le triplet inconsistant contenant c_{ins}

- RC-2 : S'il existe un concept père c_2 qui est le père de c_{ins} mais qu'il n'existe pas une relation entre le concept c_2 et la propriété p (i.e. la relation (c_2, p)) dans les deux versions O_1 et O_2, Alors supprimer le triplet inconsistant (e.g. supprimer $(c_{ins}\ p\ v\ .)$)

Si $\exists c_2 : c_2 \in O_1$, $c_2 \in O_2$, $(c_{ins}, c_2) \in HC_{01}$ et $\neg\exists$relation$(c_2,p) : p \in PO_2$ Alors supprimer le triplet inconsistant contenant la paire (c_{ins}, p)

- RC-3 : S'il existe un concept père c_2 qui est le père de c_{ins} et aussi une relation entre le concept c_2 et la propriété p (i.e. la relation (c_2, p)) dans les deux versions O_1 et O_2, Alors remplacer c_{ins} par c_2 dans le triplet inconsistant contenant la paire c_{ins} et p (e.g. remplacer $(c_{ins}$ p v .) par $(c_2$ p v .))

  Si $\exists c_2 : c_2 \in O_1$, $c_2 \in O_2$, $(c_{ins}, c_2) \in HC_{01}$ et $\exists$relation$(c_2,p) : p \in PO_2$ Alors remplacer le triplet $(c_{ins}$ p v .) par $(c_2$ p v .)

- RC-4 : S'il existe un concept c_x dans l'ontologie O_2 dont les voisins (c.f. section 4.2) (i.e. les vosins de c_x est l'ensemble des concepts, des propriétés et des instances étant en relation avec c_x) sont identiques à ceux du concept c_{ins} dans l'ontologie O_1 Alors remplacer c_{ins} par c_2 dans le triplet inconsistant contenant la paire c_{ins} et p (e.g. remplacer $(c_{ins}$ p v .) par $(c_x$ p v .))

  Si $\exists c_x : c_{ins} \in O_1$, $c_x \in O_2$ et voisins(c_{ins})= voisins(c_x) Alors remplacer le triplet $(c_{ins}$ p v .) par $(c_x$ p v .)

Note : Les règles ci-dessus concernent le cas où le concept inconsistant c_{ins} *est le domaine d'une propriété* p. *L'autre cas où* c_{ins} *est le co-domaine d'une propriété* p, *est traité de manière similaire.*

```
1.(r1 plays_role v1 .)
  (r1 type Actor .)(v1 type Role_in_the_CoP .)
2.(r2 performs_activity v2 .)
  (r2 type Role_in_the_CoP .) (v2 type Activity .)
3.(r3 performs_activity v3 .)
  (r3 type Role_in_the_CoP .) (v3 type Activity .)
4.(r4 coordinate v4 .)
  (r4 type Coordinator.)(v4 type Community_of_Practice.)
5.supprimé
6.(r6 request_by v6 .)
  (r6 type Activity .) (v6 type Actor .)
```

Figure 56 - Exemple des triplets corrigés consistants

Revenons à l'exemple ci-dessus (c.f. Figure 45): après avoir détecté les inconsistances, nous trouvons que les triplets 2, 3, 5 et 6 sont dans un état inconsistant. Sachant que le changement DeleteConcept(Member_role) entraîne la perte des liens des ressources dans les triplets 3 et 5 vers les termes correspondants dans l'ontologie, le système peut appliquer les règles ci-dessus pour corriger les triplets inconsistants. La règle RC-3 va remplacer le nom de la ressource du type Member_role par le nom du type de concept père

Role_in_the_CoP dans le triplet à la ligne 3 (c.f. Figure 56) parce que le lien entre la propriété performs_activity et le concept Role_in_the_CoP est encore gardé. La règle RC-2 va supprimer le triplet à la ligne 5 (c.f. Figure 56) parce qu'il n'existe pas de lien entre la propriété coordinate et le concept Role_in_the_CoP qui est le père du concept Facilitator dans la nouvelle ontologie O_2. La règle RC-4 détecte que les voisins du concept ACTIVITY dans l'ancienne ontologie O_1 sont identiques à ceux du concept Activity dans la nouvelle ontologie O_2, les triplets contenant la ressource de type ACTIVITY sont mis à jour : les ressources de type ACTIVITY sont remplacées par d'autres de type Activity.

```
1     Algorithm. FindingClosestConcept(newO,A,c)
2     Input: new ontology newO, set of annotations A,
3     inconsistent concept c which is currently used in A
4     Output: list LC of closest concepts of c in new
5     ontology
6     Begin
7           Load newO, A
8           /*get list P of all properties related to the
9           concept c in annotations A*/
10          P(c) ← {p1, p2,..pn}A

11          /*for each pi, find concept domain/range of pi
12          in newO then add these concepts to LC */
13          For Each pi in P(c) Do
14                tmp := getConceptDomainRange(pi, newO)
                  tmp → LC
15          EndFor
16          /*re-sort list LC, concepts having more
17          property
18          links will be given priority at the top of LC*/
19          re-sort(LC)

20          /*add others less priority concepts of newO to
21          the end of list LC*/
22          complete(LC)
23    End
```

Figure 57 - Algorithme de recherche d'un « meilleur » concept

D'autre part, nous avons toujours la possibilité de corriger les triplets inconsistants d'une manière *semi-automatique* avec l'aide de l'utilisateur. Nous avons présenté dans la section précédente ce genre de résolution dans le cas où on connaît les changements effectués. Cependant, dans le cas contraire, un problème surgit pour l'opération de remplacement : en effet, il faut pouvoir déterminer le choix d'un élément correct de remplacement. A cause de l'absence de la trace d'évolution, nous ne pouvons pas savoir comment les changements de l'ontologie ont été effectués et quels sont les résultats souhaités de ces changements. Il existe certaines recherches sur le problème de détection et d'identification des changements ontologiques mais les résultats restent encore préliminaires : un travail plus récent [Klein, 2004] a proposé des heuristiques pour comparer deux versions d'une ontologie afin de déterminer les changements effectués. D'après nous, ces approches ont encore des limites car un résultat de détection d'un changement peut avoir pour cause divers changements possibles. Par exemple, si le résultat conclut qu'il y a disparition du concept c dans la nouvelle version de

l'ontologie, cela peut correspondre soit à une suppression, soit à un renommage du concept c, soit à une fusion d'un concept c avec un autre concept, etc. Concernant ce problème, dans notre approche, chaque règle de détection d'inconsistance nous donne également un résultat de changement détecté qui peut être correspondre à différents changements.

Au lieu de déterminer quels changements ont été effectués, notre approche vise à proposer à l'utilisateur les meilleures manières de corriger les annotations sémantiques inconsistantes. Nous développons un mécanisme pour choisir le *"meilleur"* élément dans la nouvelle version de l'ontologie pour remplacer celui qui devient obsolète dans l'annotation. La Figure 58 représente un exemple d'un algorithme pour trouver le concept le plus proche (c.f. le "meilleur") du concept inconsistant c dans la nouvelle ontologie newO pour remplacer le concept obsolète c dans l'annotation A.

```
1     <rules>
2        <rule id="R-1"
             handler="fr.inria.acacia.coswem.model.impl.-
             ConceptRuleHandler"
3            name="check for the existence of concept">
4            <query type="concept">
5            < . . .>
6            <queryString>
7                  <. . .>
8            </queryString>
9            </query>
10           <param key="INCONSISTENY_XSL" value="WEB-
11           INF/coswem-withouttrace/xsl/concept.xsl"/>
12
13           <corrector name="rename concept"
14           handler="fr.inria.acacia.coswem.model.impl.-
15           RenameConceptCorrector"/>
16           <corrector name="remove concept"
17           handler="fr.inria.acacia.coswem.model.impl.-
18           RemoveConceptCorrector"/>
19           <. . .>
20    </rule>
```

Figure 58 - Représentation d'une règle de correction d'inconsistance

Représentation des règles de correction d'inconsistance

Comme les règles de détection d'inconsistance, nous représentons également les règles de correction d'inconsistance à l'aide d'un fichier XML. Cela nous permet facilement de formuler et de modifier les règles selon notre besoin (c.f. Figure 58). Dans ces règles de correction, nous intégrons des balises (i.e. tags) nommées <corrector> pour spécifier les opérations à appliquer en vue de corriger des annotations inconsistantes. Ces opérations sont divisées en deux types principaux : (i) supprimer ou (ii) remplacer un élément du triplet. La Figure 58 illustre cela par un exemple qui permet de corriger des ressources inconsistantes du type concept dans les triplets. Nous pouvons également ajouter dans ces règles

des paramètres qui sont utilisés pour les traitements techniques dans le processus de correction des inconsistances (e.g. le feuille de représentation XSLT).

5.4 Jeu de tests et exemples

Nous avons présenté dans les sections précédentes les approches qui nous permettent de détecter et de corriger des annotations sémantiques. Cette section va, quant à elle, se pencher sur tous les cas possibles apparus dans une annotation sémantique, et particulièrement sur les cas potentiels qui pourraient influencer l'état consistant d'une annotation. Nous rassemblons ces cas possibles dans un jeu de tests qui est validé dans le composant de gestion de l'évolution du système CoSWEM. D'autre part, nous présentons également des exemples de triplets équivalents à chaque cas du jeu de tests afin d'illustrer chaque test et son résultat détecté souhaité.

Jeu de tests

Nous utilisons la notion du modèle de triplet en RDF (`c p v.`) pour exprimer l'utilisation d'un triplet dans une annotation sémantique. Les éléments du triplet sont les ressources annotées des types définis dans l'ontologie de base. Par exemple, la ressource `c` est du type de concept `C`, la ressource `v` est peut-être du type de concept `O` (ou du type de valeur littérale `V`), ces ressources sont reliées par la propriété `p` (avec `C, O, V, p` sont déjà définis dans l'ontologie). L'utilisation des éléments du triplet peut être correcte ou incorrecte par rapport au modèle de l'ontologie de base. Reprenons l'exemple ci-dessus, si la ressource `c` du triplet est du type `C`, donc `c` est correct. Mais si la ressource `c` est d'un autre type `Cx` dans l'annotation, alors `c` est incorrect. Le Tableau 9 présente le jeu de tests contenant tous les tests de l'utilisation des triplets qui sont intégrés dans le mécanisme de détection d'inconsistance du système CoSWEM.

Tableau 9 - Jeu de tests pour les annotations sémantiques

No.	Description du cas de test	Résultat détecté souhaité
1	Triplet utilise une instance du type correct d'un concept	- Pas d'inconsistance
2	Triplet utilise une instance du type incorrect d'un concept	- Inconsistant
3	Triplet (`c p v .`), tous les éléments sont corrects	- Pas d'inconsistance
4	Triplet (`c p v .`), `p` et `v` sont corrects	- Concept inconsistant

	mais c est incorrect	- Domaine inconsistant
5	Triplet (c p v .), c et p sont corrects mais v est incorrect	- Co-domaine inconsistant - Concept inconsistant (si v est du type concept)
6	Triplet (c p v .), c et v sont corrects mais p est incorrect	- Propriété inconsistante
7	Triplet (c p v .), c est correct mais p et v sont incorrects	- Propriété inconsistante - Concept inconsistant (si v est du type concept)
8	Triplet (c p v .), p est correct mais c et v sont incorrects	- Concept inconsistant - Domaine inconsistante - Co-domaine inconsistante (si v est du type concept)
9	Triplet (c p v .), v est correct mais c et p sont incorrects	- Concept inconsistant - Propriété inconsistante
10	Triplet (c p v .), c , p et v sont incorrects	- Concept inconsistant - Propriété inconsistante
11	Type des données (p, v) v est défini comme un concept dans l'ontologie, mais portant une valeur littérale dans l'annotation	- Type de données inconsistant
12	Type des données (p, v) v est défini comme une valeur littérale dans l'ontologie, mais portant un type de concept dans l'annotation	- Type de données inconsistant

Exemples du jeu de tests

1. Cas 1 : Triplet utilise une instance correcte de l'ontologie

<oc:actors rdf:about="http://www.inria.fr/acacia/ontocreator#ACT_0"/>

<oc:Role_in_the_CoP rdf:about="http://www.inria.fr/acacia/ontocreator#roleCoP_0"/>

- Description: Les concepts `actors` et `role_in_the_CoP` de l'ontologie ont des instances `ACT_0, roleCoP_0` respectivement. Ces instances sont utilisées dans les triplet de l'annotation.

- Résultat détecté souhaité: Pas d'inconsistance

2. Cas 2 : Triplet utilise une instance incorrecte de l'ontologie

<oc:actors1 rdf:about="http://www.inria.fr/acacia/ontocreator#ACT_1"/>

<oc:actors2 rdf:about="http://www.inria.fr/acacia/ontocreator#ACT_2"/>

- Description : Les concepts `actors1, actors2` n'existent pas dans l'ontologie. L'annotation contient des ressources du type de ces concepts.

- Résultat détecté souhaité: inconsistance sur le concept

3. Cas 3 : Tous les éléments du triplet (c p v .) sont corrects par rapport à l'ontologie de base

<oc:Role_in_the_CoP rdf:about="http://www.inria.fr/acacia/ontocreator#roleCoP_0"/>

<oc:actors rdf:about="http://www.inria.fr/acacia/ontocreator#ACT_0">
<oc:plays_role rdf:resource="http://www.inria.fr/acacia/ontocreator#roleCoP_0"/>
</oc:actors>

- Description: Le triplet contient des ressources de bons types. Les concepts `Role_in_the_CoP, actors` et la propriété `plays_role` sont déjà définis dans l'ontologie.

- Résultat détecté souhaité: Pas d'inconsistance.

4. Cas 4 : Triplet (c p v .) dans lequel p et v sont corrects, c est incorrect

<oc:Role_in_the_CoP rdf:about="http://www.inria.fr/acacia/ontocreator#roleCoP_0"/>

<oc:actors1 rdf:about="http://www.inria.fr/acacia/ontocreator#ACT_0">
<oc:plays_role rdf:resource="http://www.inria.fr/acacia/ontocreator#roleCoP_0"/>
</oc:actors1>

- Description: Le concept `Role_in_the_CoP` et la propriété `plays_role` sont déjà définis dans l'ontologie mais le concept `actors1` n'est pas encore défini.

- Résultat détecté souhaité: inconsistance sur le concept et sur le domaine.

5. Cas 5 : Triplet (c p v .) dans lequel c et p sont corrects, v est incorrect

<oc:community_of_practice1 rdf:about="http://www.inria.fr/acacia/ontocreator#CoP_0"/>

```
<oc:Coordinator rdf:about="http://www.inria.fr/acacia/ontocreator#Coor_0">
        <oc:coordinate rdf:resource="http://www.inria.fr/acacia/ontocreator#CoP_0"/>
</oc:Coordinator>
```

- Description: Le concept `Coordinator` et la propriété `coordinate` sont déjà définis dans l'ontologie mais le concept `community_of_practice1` n'est pas encore défini.

- Résultat détecté souhaité: inconsistance sur le co-domaine ou sur le concept (si `v` est du type concept).

6. Cas 6 : Triplet (c p v .) dans lequel c et v sont corrects, p est incorrect

```
<oc:actors rdf:about="http://www.inria.fr/acacia/ontocreator#ACT_0">

<oc:community_of_practice rdf:about="http://www.inria.fr/acacia/ontocreator#CoP_2">
        <oc:has_actor_0 rdf:resource="http://www.inria.fr/acacia/ontocreator#ACT_0"/>
</oc:community_of_practice>
```

- Description: Les concepts `actors` et `community_of_practice` sont déjà définis dans l'ontologie mais la propriété `has_actors_0` n'est pas définie.

- Résultat détecté souhaité: inconsistance sur la propriété.

7. Cas 7 : Triplet (c p v .) dans lequel c est correct, p et v sont incorrects

```
<oc:actors2 rdf:about="http://www.inria.fr/acacia/ontocreator#ACT_2"/>

<oc:community_of_practice rdf:about="http://www.inria.fr/acacia/ontocreator#CoP_2">
        <oc:has_actor_2 rdf:resource="http://www.inria.fr/acacia/ontocreator#ACT_2"/>
</oc:community_of_practice>
```

- Description: Le concept `community_of_practice` est défini dans l'ontologie mais la propriété `has_actors_2` et le concept `actors2` ne sont pas définis.

- Résultat détecté souhaité: inconsistance sur la propriété, inconsistance sur le concept (si `v` est du type concept).

8. Cas 8 : Triplet (c p v .) dans lequel p est correct, c et v sont incorrects

```
<oc:actors2 rdf:about="http://www.inria.fr/acacia/ontocreator#ACT_2"/>

<oc:community_of_practice_0 rdf:about="http://www.inria.fr/acacia/ontocreator#CoP_2">
        <oc:has_actor rdf:resource="http://www.inria.fr/acacia/ontocreator#ACT_2"/>
</oc:community_of_practice_0>
```

- Description: La propriété `has_actor` est déjà définie dans l'ontologie mais les concepts `community_of_practice_0` et `actors2` ne sont pas définis.

- Résultat détecté souhaité: inconsistance sur le concept, sur le domaine et sur le co-domaine (si `v` est du type concept).

9. Cas 9 : Triplet (c p v .) dans lequel v est correct, c et p sont incorrects

```
<oc:domaine rdf:about="http://www.inria.fr/acacia/ontocreator#domaine_0"/>

<oc:community_0 rdf:about="http://www.inria.fr/acacia/ontocreator#CMT_0">
        <oc:has_domain_2 rdf:resource="http://www.inria.fr/acacia/ontocreator#domaine_0"/>
</oc:community_0>
```

- Description: Le concept `domaine` est déjà défini dans l'ontologie mais le concept `community_ 0` et la propriété `has_domaine2` ne sont pas définis.

- Résultat détecté souhaité: inconsistance sur le concept et sur la propriété.

10. Cas 10 : Tous les éléments du triplet (c p v .) sont incorrects par rapport à l'ontologie de base

```
<oc:actors2 rdf:about="http://www.inria.fr/acacia/ontocreator#ACT_2"/>

<oc:ACTIVITY rdf:about="http://www.inria.fr/acacia/ontocreator#AC_tmp_0">
        <oc:order_by rdf:resource="http://www.inria.fr/acacia/ontocreator#ACT_2"/>

</oc:ACTIVITY>
```

- Description: Les concepts `ACTIVITY`, `actors2` et la propriété `order_by` ne sont pas définis.

- Résultat détecté souhaité: inconsistance sur le concept et sur la propriété.

11. Cas 11 : Type des données (p, v) v est le concept dans l'ontologie, mais portant une valeur littérale dans l'annotation

```
<oc:actors2 rdf:about="http://www.inria.fr/acacia/ontocreator#ACT_2"/>

<oc:objectif rdf:about="http://www.inria.fr/acacia/ontocreator#OBJ_0">
        <oc:new_objective rdf:resource="http://www.inria.fr/acacia/ontocreator#ACT_2"/>
</oc:objectif>
```

- Description: La relation `[objectif]-(new_objective)-[Literal]` est définie dans l'ontologie. La propriété prend une valeur littérale.

- Résultat détecté souhaité: inconsistance sur le type de données.

12. Cas 12 : Type des données (p, v) v est une valeur littérale dans l'ontologie, mais portant un type de concept dans l'annotation

```
<oc:ACTIVITY rdf:about="http://www.inria.fr/acacia/ontocreator#AC_tmp_0">
        <oc:order_by>Etudiant</oc:order_by>
</oc:ACTIVITY>
```

- Description: La relation `[ACTIVITY]-(order_by)- [actors]` est définie dans l'ontologie. La propriété prend une valeur du type de concept.

- Résultat détecté souhaité: inconsistance sur le type de données.

5.5 Conclusion

Après les travaux sur l'évolution de l'ontologie, nous continuons à étudier les conséquences des changements de l'ontologie sur les annotations sémantiques qui utilisent les concepts et les propriétés définies dans l'ontologie. Les annotations devenant inconsistantes ont besoin d'évoluer pour pouvoir s'adapter aux changements de l'ontologie.

Les tâches principales de l'évolution de l'annotation sémantique sont de détecter les annotations inconsistantes et de corriger ces inconsistances en vue d'assurer la consistance globale de toute la base d'annotations ainsi que l'utilisation cohérente de l'ontologie et des annotations ensembles. Nous avons présenté dans ce chapitre les méthodes permettant de gérer l'évolution de l'annotation sémantique et particulièrement de résoudre le problème de détection et de résolution d'inconsistance pour les annotations.

Nous nous intéressons à deux scénarios de l'évolution de l'ontologie du Web sémantique dans lesquels nous pouvons garder ou ne pas garder une trace de changements ontologiques effectués. Ces deux scénarios arrivent souvent en réalité et ils affectent également l'évolution de l'annotation sémantique. Par conséquent, nous avons développé et intégré dans le système de gestion de l'évolution CoSWEM deux approches : (i) une approche procédurale et (ii) une approche basée sur des règles qui sont équivalentes à deux scénarios d'évolution : avec trace et sans trace de changement de l'ontologie. Ces deux approches ont été validées par plusieurs cas.

Dans le chapitre suivant, nous allons présenter en détail les résultats ainsi que l'évaluation de notre approche de recherche. Nous décrivons également les phases de l'implémentation du système de gestion de l'évolution CoSWEM ainsi que les études de cas qui sont appliquées et validées dans ce système.

Partie III

Exploitation et évaluation du système CoSWEM

Chapitre 6

Implémentation et Évaluation du CoSWEM

Nous avons présenté dans les chapitres précédents notre approche sur la gestion de l'évolution d'un Web sémantique d'entreprise. Dans le chapitre 4, nous avons examiné le processus d'évolution de l'ontologie avec deux scénarios d'évolution principaux : avec trace et sans trace des changements de l'ontologie. Dans le chapitre 5, deux approches ont été présentées : une approche procédurale puis une approche basée sur des règles. Ces approches sont utilisées pour la détection et la résolution des inconsistances des annotations sémantiques, suite à une ou plusieurs modifications de l'ontologie. Ces deux approches correspondent chacune à un des deux scénarios d'évolution. Nous avons construit un système de gestion de l'évolution intitulé CoSWEM qui a pour but d'aider l'utilisateur à réaliser d'une manière semi-automatique les tâches de détection et de résolution des inconsistances générées au cours de l'évolution. Dans ce chapitre, nous présentons tout d'abord le processus d'implémentation du système CoSWEM, ainsi que l'architecture et la conception du système. Ensuite, nous décrivons l'expérimentation et le processus de validation du système sur deux projets. Nous évaluons finalement les résultats acquis à travers les projets expérimentés et nous conclurons ce chapitre.

6.1 Implémentation du système

Dans cette section, nous présentons l'architecture d'implémentation du système CoSWEM et la conception de ce système. Nous illustrons les fonctionnalités principales du système à l'aide d'exemples d'interfaces graphiques.

6.1.1 Architecture d'implémentation

Comme le montrait l'architecture du système CoSWEM dans le chapitre 3 (c.f. Figure 15), ce système fait intervenir différents types d'agents humains : (i) l'ontologiste (fournisseur de l'ontologie) qui fournit et modifie l'ontologie utilisée dans le système, (ii) l'annotateur (ingénieur d'annotation) qui crée et modifie la base d'annotations utilisant l'ontologie fournie, (iii) l'ingénieur du système qui gère les fonctions du système et (iv) l'utilisateur normal qui utilise et exploite le système (il pourrait être représenté pour chaque type d'utilisateur ci-dessus).

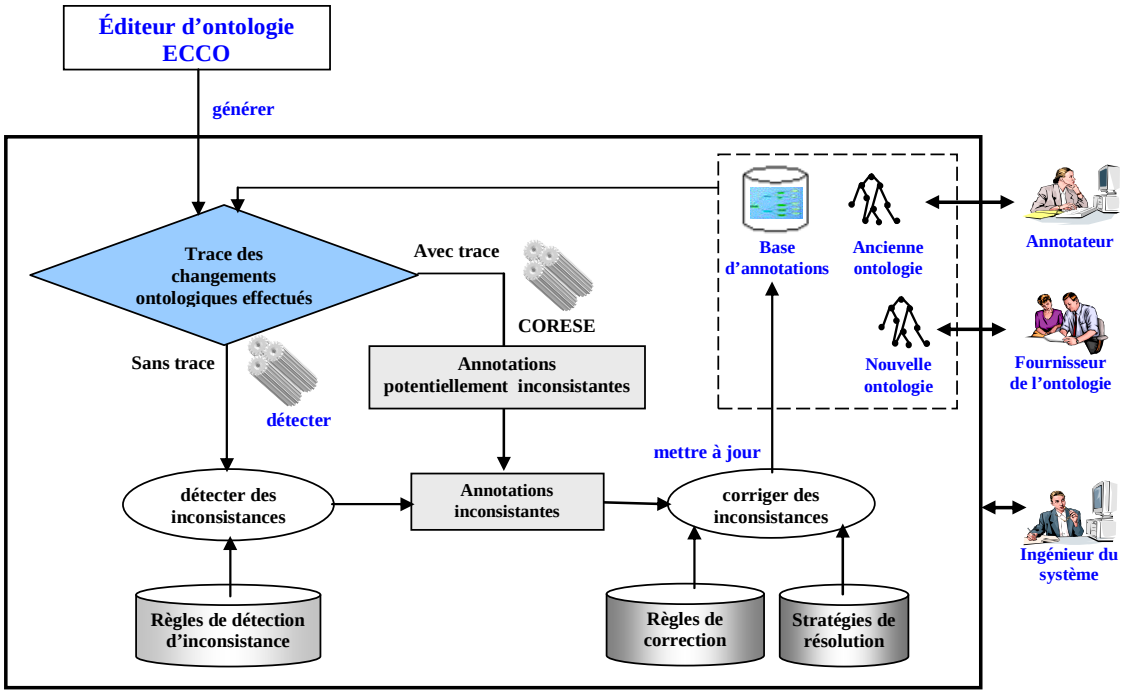


Figure 59 - Architecture d'implémentation du système CoSWEM

Ces agents interagissent avec le système à travers les processus principaux suivants (c.f. Figure 59):

- CoSWEM est intégré dans un système de gestion des connaissances du Web sémantique d'entreprise, il comporte alors différentes ontologies (i.e. l'ancienne ontologie, la nouvelle ontologie…) et une base d'annotations qui décrit les ressources de l'entreprise en reposant sur les concepts et relations définis dans l'ancienne ontologie.

- L'ontologiste modifie l'ancienne ontologie en utilisant un éditeur d'ontologie (e.g. l'outil ECCO) et crée ainsi une nouvelle version de l'ontologie. Les changements ontologiques effectués sont capturés dans une trace d'évolution (ou trace de changements) qui est générée automatiquement (i.e. trace d'évolution générée par ECCO) par l'éditeur de l'ontologie après la modification. Ces changements peuvent affecter la consistance des autres parties dépendantes de l'ancienne ontologie et particulièrement de la base d'annotations concernée. La trace d'évolution peut être gardée, ou au contraire dans d'autres cas, on peut ne disposer d'aucune information sur les changements effectués.

- Jusqu'à cette étape, CoSWEM vérifie si la trace de changements ontologiques existe ou pas. Les deux scénarios possibles correspondent aux deux cas suivants : la trace d'évolution est conservée ou on ne garde pas cette trace. CoSWEM intègre le moteur de recherche sémantique Corese qui lui permet de lancer des requêtes selon différentes exigences sur la hiérarchie des concepts et des propriétés de l'ontologie ainsi que sur la base d'annotations. Le moteur Corese facilite la recherche des éléments affectés et obsolètes par rapport aux changements ontologiques effectués.

- La détection des annotations sémantiques inconsistantes est réalisée avec l'aide du moteur de recherche Corese et certaines fonctions intégrées dans CoSWEM. Si les changements effectués sont connus (i.e. avec trace), le système récupère d'abord les annotations potentiellement inconsistantes qui peuvent inclure des annotations consistantes et d'autres inconsistantes. Il cherche ensuite des annotations vraiment inconsistantes en vérifiant les types de changements conservés dans la trace d'évolution (e.g. les changements entraînant la *correction obligatoire* qui entraînent les annotations sémantiques inconsistantes). Au contraire, si la trace d'évolution n'est pas gardée, il applique des règles de détection d'inconsistance pour chercher les annotations qui deviennent obsolètes et inconsistantes par rapport à la nouvelle version de l'ontologie.

- Quant à la résolution des annotations inconsistantes détectées, nous pouvons la réaliser d'une manière automatique ou semi-automatique. Dans le cas où l'on connaît les changements de l'ontologie qui ont eu lieu, nous connaissons également les stratégies de résolution choisies et appliquées pour traiter le changement correspondant. Le système peut utiliser des règles de correction afin de réaliser automatiquement la stratégie de résolution choisie pour l'annotation sémantique qui a été construite respectivement pour chaque stratégie de l'ontologie. D'autre part, si le système n'a aucune information sur l'évolution de l'ontologie, il ne sait alors pas comment choisir automatiquement la stratégie de résolution convenable pour corriger l'inconsistance. Dans ce cas, nous pouvons réaliser la tâche de résolution des annotations inconsistantes

d'une manière semi-automatique, c.-à-d. avec l'aide de l'utilisateur. Le système propose alors à l'utilisateur de choisir une opération qui semble lui convvenir parfaitement (i.e. supprimer le triplet posant problème, remplacer le triplet problématique, laisser sans modification…).

- Finalement, la base d'annotations est mise à jour avec les nouvelles annotations sémantiques consistantes par rapport à la nouvelle version de l'ontologie. Cela garantit l'utilisation consistante globale de cette base d'annotations.

6.1.2 Conception du système

En vue du déploiement sur le web du système CoSWEM afin de pouvoir l'appliquer au Web sémantique d'entreprise, nous avons choisi le modèle MVC (Model-View-Controller) pour développer CoSWEM. L'application est développée en Java, JSP/Servlet et d'autres langages de représentation tels que HTML, XML… Elle est déployée dans l'environnement du Web utilisant le serveur Web Tomcat d'Apache.

Modèle MVC global

L'architecture MVC (Modèle-Vue-Contrôleur) est un motif de conception qui est très utilisé de nos jours dans de nombreux domaines et langages. Ce modèle est dédié au développement d'applications logicielles en séparant le modèle de données, l'interface utilisateur et la logique de contrôle. Un avantage du modèle MVC permet d'avoir non seulement des objets réutilisables pour d'autres applications, mais aussi de pouvoir faire évoluer aisément son programme. Ainsi, si l'on souhaite modifier sa base de données, il suffit de revoir son "modèle" et cela est aussi valable dans le cas où l'on souhaite changer d'interface. Les 3 parties du model MVC sont réellement autonomes.

Le modèle de conception impose donc une séparation en 3 couches :

- *Model (Modèle) :* représente le comportement de l'application tel que le traitement des données, l'interaction avec la base de données, etc.

- *Controller (Contrôleur) :* prend en charge l'interface entre le modèle et le client, et aussi la gestion des événements de synchronisation pour mettre à jour la vue ou le modèle.

- *View (Vue) :* correspond à l'interface avec laquelle l'utilisateur interagit. Elle n'effectue aucun traitement, elle se contente simplement d'afficher les données que lui fournit le modèle. Il peut tout à fait y avoir plusieurs vues qui présentent les données d'un même modèle.

La Figure 60 représente le modèle MVC global et le diagramme de séquence d'un modèle MVC simple . La couche de contrôleur analyse la requête, et éventuellement extrait les informations dans l'URL de la requête. Elle fait ensuite une mise à jour du modèle si nécessaire, en lui passant des paramètres, détermine la vue à afficher et demande son affichage.

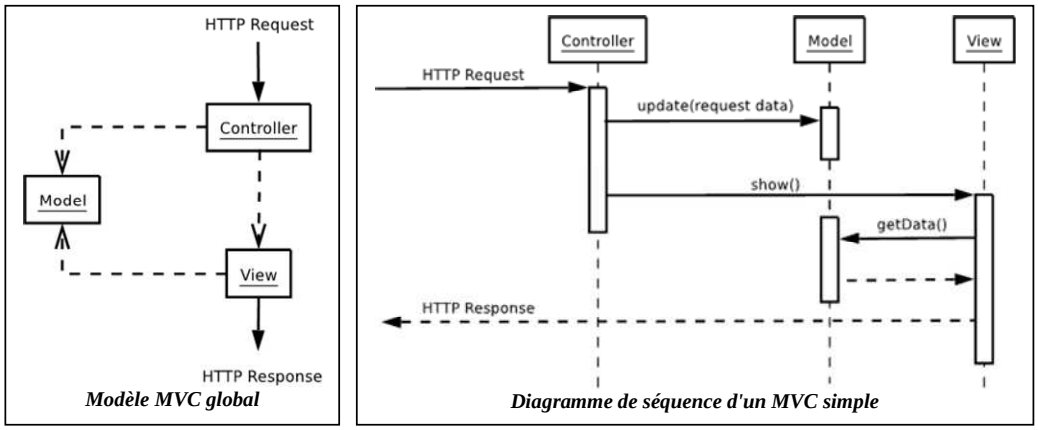


Figure 60 - Modèle global et diagramme de séquence du MVC

Modèle MVC appliqué dans CoSWEM

Nous appliquons le modèle MVC en trois couches à la conception du système CoSWEM (c.f. Figure 61) :

- *La couche Modèle :* Cette couche sert à décrire les formats de règles et de stratégies de résolution pour l'ontologie et l'annotation sémantique. Elle contient des classes pour gérer des règles de détection d'inconsistance (i.e. règles sur le concept, sur la propriété, sur le domaine/co-domaine et sur le type des données). Elle définit les méthodes d'accès et les "Beans" qui sont dédiées aux tâches spécifiques telles que la gestion des règles, la gestion des requêtes de l'utilisateur, la gestion des correcteurs pour corriger les annotations inconsistantes, etc. Elle comporte également la classe DOM permettant la manipulation de fichiers XML (et donc de fichiers d'annotations). Cette couche se base sur une API des librairies Corese et Semtag qui facilitent la manipulation des opérations nécessaires sur l'ontologie et sur l'annotation sémantique.

- *La couche Vue :* Le format de représentation essentiel est HTML qui est affiché par le navigateur du Web. Cette couche se compose des transformateurs et des templates XSL qui nous permettent de transformer les résultats et les données en XML vers HTML pour qu'ils puissent être renvoyés sous une forme lisible à l'utilisateur.

- *La couche Contrôleur :* La partie principale de cette couche est la Façade CoSWEM qui a pour but de recevoir des requêtes de l'utilisateur, d'analyser les requêtes et ensuite de transférer des requêtes avec les

paramètres nécessaires aux fonctions correspondantes à la couche modèle. Cette couche contient aussi des servlets qui sont dédiées à la synchronisation du modèle et de la vue. Elle reçoit tous les événements de l'utilisateur et enclenche les actions correspondantes à effectuer.

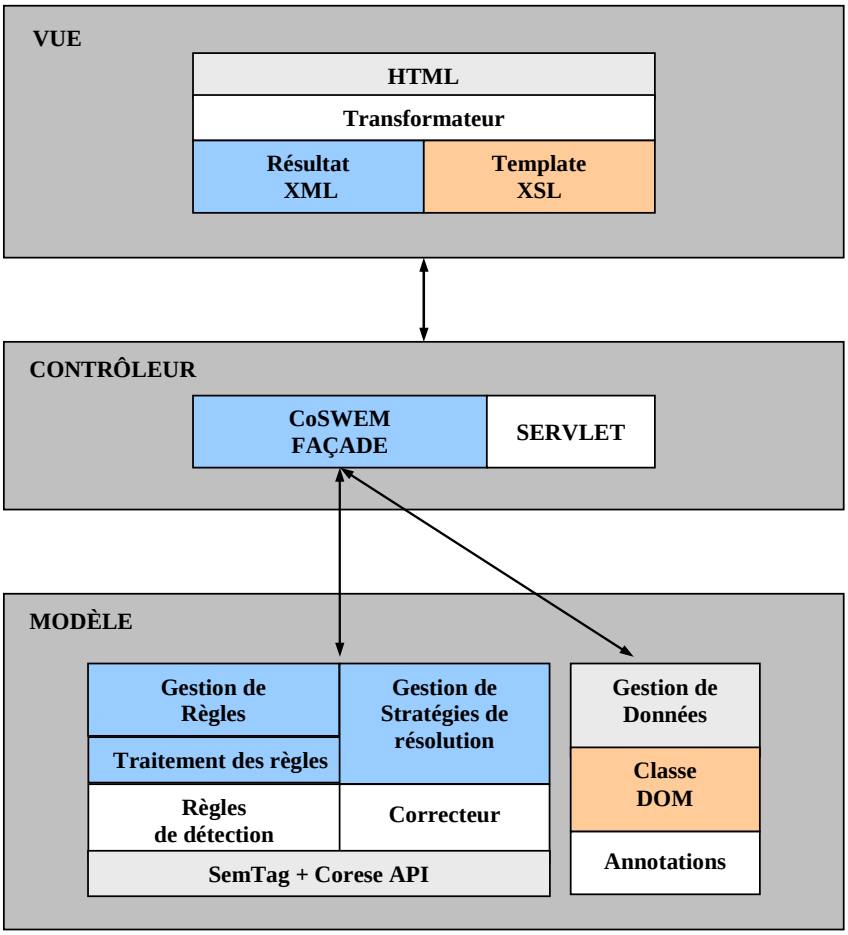


Figure 61 - Modèle MVC appliqué dans CoSWEM

Conception des règles

Dans le chapitre 5, nous avons présenté les règles de détection et de correction d'inconsistance sur les annotations sémantiques influencées par la modification de l'ontologie. La détection d'inconsistance est basée sur des règles et consiste à vérifier des règles sur le concept, sur la propriété, sur le domaine/co-domaine et sur le type des données.

La Figure 62 introduit la conception des règles implémentées dans le système CoSWEM. Ces règles sont représentées sous forme XML. Chaque règle comporte d'une part une requête en SPARQL et d'autre part les classes pour permettre de réaliser des tâches correspondantes demandées dans les requêtes. Nous avons implémenté un parseur de règles permettant d'extraire les données et les informations nécessaires qui sont conservées dans les fichiers de description

des règles (en XML). Ce parseur analysera une projection de règles (i.e. RuleMapping) qui projette entre les classes et les tâches à résoudre correspondantes.

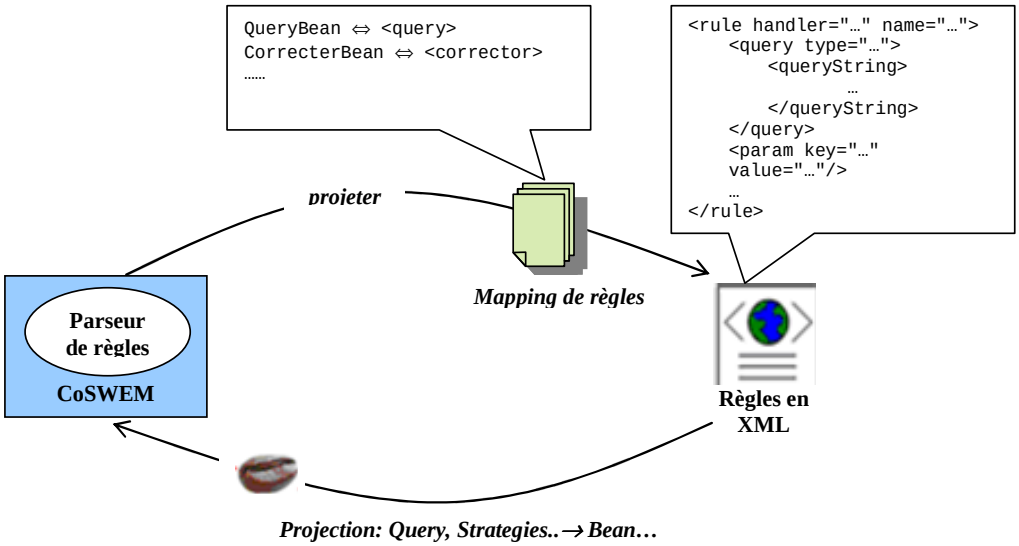


Figure 62 - Conception des règles

Nous avons donné un exemple de représentation des règles en XML (c.f. Figure 55, section 5.3.2). Dans la section suivante, nous décrivons la structure du ficher de règles et les balises (i.e. tags) utilisées pour représenter ces règles (c.f. Tableau 10).

Tableau 10 - Description des tags XML du ficher de règles (rule.xml)

Nom du Tag	Description
`<rules>`	Tag de début du fichier de règles
`<rule>`	Tag qui définit l'identificateur de chaque règle, par exemple la règle sur le concept, sur la propriété…
`<name>`	Nom de la règle
`<handler>`	Tag qui définit la classe Java correspondante pour gérer la règle
`<description>`	Description du tag
`<query>`	Tag qui définit la requête qu'on lance au moteur de recherche Corese. Il y a des types de requêtes sur le concept, la propriété, le triplet, le type de l'instance…

<queryString>	Tag qui décrit le contenu de la requête
<param>	Description des paramètres nécessaires, par exemple le template XLS, le chemin du ficher de configuration…
<corrector>	Tag qui définit la stratégie de résolution pour corriger l'inconsistance.

D'autre part, le parseur de règles utilise également un fichier de projection de règles qui définit les tags en XML. Ce fichier sert à établir une correspondance entre les données dans le fichier de description des règles et les Beans dont le rôle est de réaliser des tâches spécifiques. La Figure 63 représente un extrait du fichier de projection des règles et ses tags décrits en XML (c.f.Tableau 11).

```
<mapping>
    <description>Rule mapping</description>
    <class
     name="fr.inria.acacia.coswem.model.rule.RuleManagerBean"
          identity="rulemanager">
          <map-to xml="rules"/>
          <field name="rules"
            type="fr.inria.acacia.coswem.model.rule.RuleBean"
                  direct="true" collection="vector">
                  <bind-xml name="rule" node="element"/>
          </field>
    </class>
</mapping>
```

Figure 63 - Extrait du fichier de projection des règles (rule-mapping.xml)

Tableau 11 - Description des tags XML du ficher de projection (rule-mapping.xml)

Nom du Tag	Description
<mapping>	Tag de début du fichier de projection des règles
<description>	Description du fichier
<class>	Tag qui définit la classe correspondante pour vérifier la règle. Une classe a plusieurs champs décrivant les tags différents du fichier "rules.xml". Le tag <class> contient des attributs suivants : - <name> : nom de la classe - <map-to xml> : nom du fichier de règles correspondant (i.e. "rules.xml") - <field name> : nom du tag correspondant dans le fichier de règles "rules.xml"
<field>	Tag qui définit les paramètres d'un champ (i.e. field) de la classe. Il a les attributs suivants : - <name> : nom du tag

	- <type> : type de données du tag - <direct> : valeur vrai ou faux - <collection> : valeur vector ou map
<bind-xml>	Nom du tag correspondant dans le fichier “rules.xml”

Application des règles et Présentation des résultats

Dans cette section, nous présentons le processus de vérification des règles et son implémentation dans CoSWEM. Lorsqu’on lance la détection des annotations sémantiques inconsistantes, toutes les règles sont vérifiées.

Chaque fois que le système CoSWEM reçoit une demande de détection d’inconsistance, le parseur des règles est démarré. Il vérifie la demande et enclenche les fonctions correspondantes qui sont construites en se basant sur les API de Corese et de SemTag. Après avoir exécuté ces fonctions, les résultats sont renvoyés sous forme de fichiers en XML. CoSWEM utilise le DOM pour lire les résultats en XML et applique des filtres XSL pour interpréter les résultats en XML les traduire en HTML pour être affichés sur une page web (c.f. Figure 64).

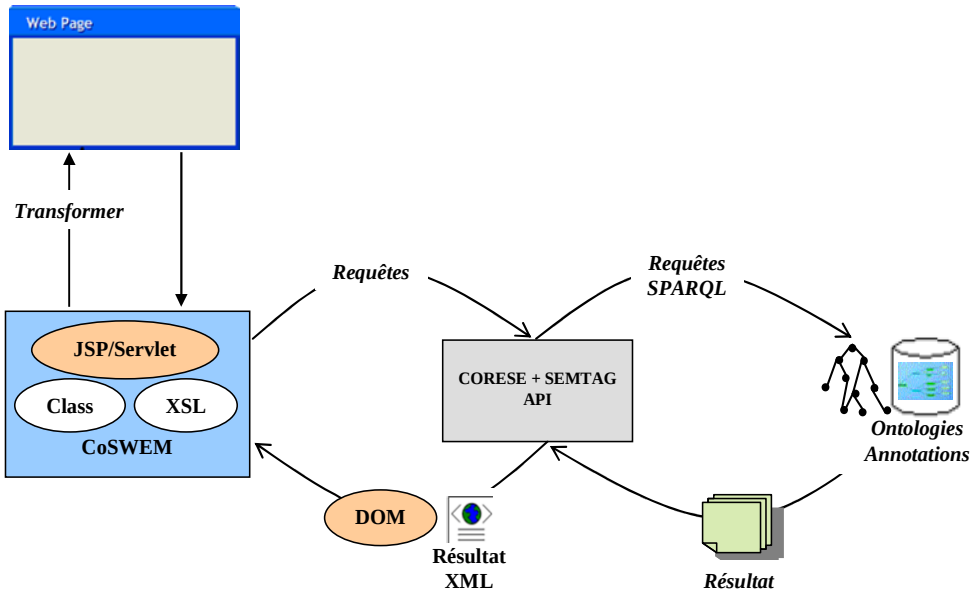


Figure 64 - Application des règles et présentation des résultats

6.2 Expérimentations

Afin de tester la fiabilité et de montrer la faisabilité de notre approche, nous avons développé le système de gestion de l’évolution CoSWEM dont les fonctionnalités principales ont été présentées dans les sections précédentes. Ce

système a besoin d'être expérimenté dans les contextes et scénarios réels afin de vérifier ses fonctions.

Dans un premier temps, nous avons fait des tests de CoSWEM et validé nos deux scénarios dans deux projets : le projet européen PALETTE et le projet ANR E-WOK_HUB. La partie suivante présente brièvement ces deux projets en soulignant leurs caractéristiques principales et leurs exigences propres qui concernent étroitement notre travail de thèse. Nous décrivons ensuite les processus d'expérimentation et évaluons les résultats acquis à travers l'expérimentation de CoSWEM sur ces deux projets.

6.2.1 Description des projets de l'expérimentation

Projet PALETTE[40]

PALETTE (Pedagogically sustained Adaptive LEarning Through the exploitation of Tacit and Explicit knowledge) est un projet européen qui vise aux objectifs suivants (i) développer de manière participative des outils et services interopérables et innovants et (ii) faciliter et augmenter l'apprentissage individuel et organisationnel dans les Communautés de Pratique (CoPs). Les CoPs sont considérés comme des groupes de personnes qui partagent un intérêt ou une passion pour quelque chose qu'ils font, et qui apprennent comment l'améliorer à travers leurs interactions régulières. Ce projet est réalisé dans le cadre de coopération de plusieurs partenaires tels que l'INRIA[41], ERCIM[42], EPFL[43] … et il se décompose en 9 lots ("workpackages") principaux.

Concernant le travail de thèse, nous nous intéressons au lot *WP3-Services de Gestion des connaissances* (c.f. Figure 65) dont l'objectif est de (i) faciliter l'accès, le partage et la réutilisation des connaissances (i.e. tacites ou explicites, individuelles ou collectives) et de (ii) générer les nouvelles connaissances liées aux activités de CoPs. Les tâches principales de ce lot sont d'élaborer des ontologies décrivant les connaissances et activités des CoPs et de développer des services de gestion des connaissances des CoPs.

[40] http://palette.ercim.org/

[41] www.inria.fr/

[42] www.ercim.org/

[43] www.epfl.ch/

Actuellement, les ontologies décrivant les connaissances du domaine et des services pour les CoPs sont construites et en train d'être utilisées. Puisque les ontologies sont élaborées à partir d'un ensemble des thésaurus contenant des termes spécialisés du domaine des CoPs, elles sont aussi développées en plusieurs phases et par différents partenaires… ces ontologies sont donc modifées et évoluent. Il existe finalement plusieurs versions différentes d'une ontologie et il y a aussi des cas où les traces de changements entre ces versions n'ont pas été bien notées. Cela nous apporte un bon scénario d'évolution de l'ontologie sans trace de changement et des données évolutives à expérimenter dans notre travail. Nous nous sommes intéressés aux différentes versions de l'ontologie O'CoP [Tifous et al., 2007] dans ce projet PALETTE avec les annotations sémantiques reposant sur cette ontologie.

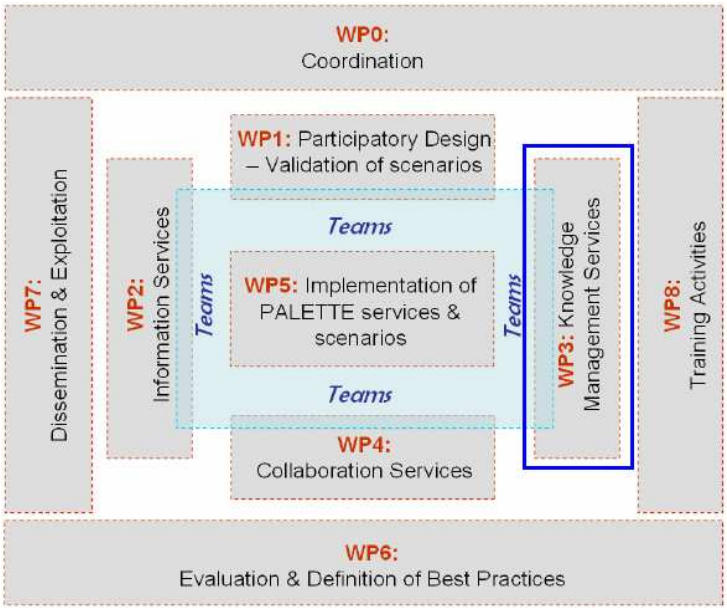


Figure 65 - Décomposition des lots du projet PALETTE

Projet E-WOK_HUB[44]

E-WOK HUB (Environmental Web Ontology Knowledge Hub) est un projet ANR RNTL[45] de trois ans entre plusieurs partenaires académiques (e.g. INRIA, ENSMP-ARMINES[46], ENSMA[47] …) et partenaires industriels (e.g. BRGM[48], EADS[49], IFP[50]). Ce projet vise à tirer bénéfice des travaux entrepris sur

[44] http://www-sop.inria.fr/acacia/project/ewok/index.html

[45] Agence Nationale de la Recherche - Réseau National des Technologies Logicielles

[46] http://www.ensmp.fr

[47] http://www.ensma.fr/

[48] http://www.brgm.fr/

[49] http://www.eads.net

le Web sémantique pour développer des systèmes opérationnels autorisant la coopération sur internet entre différentes organisations (entreprises, instituts, ...) impliqués dans un workflow d'ingénierie. Les objectifs spécifiques de ce projet sont de mettre en place un ensemble de portails communicants (i.e. les Hubs e-WOK), proposant à la fois (i) des applications web accessibles aux utilisateurs finaux à travers des IHM en ligne (ii) des services web accessibles aux applications à travers des interfaces programmatiques. La Figure 66 présente un ensemble de portails web (intégrés dans serveur web sémantique) offrant un accès sémantique à des ressources documentaires, des données métiers et des services (génériques ou métiers) en utilisant des ontologies adéquates. Nous nous intéressons à certains services tels que la gestion des ontologies, la gestion d'une mémoire de projet, l'aide à l'annotation… Au niveau applicatif, ce projet se focalise sur la veille technologique et sur la caractérisation des sites de stockage du CO2 et les applications à d'autres domaines thématiques dans le domaine des géosciences (risques naturels, eau souterraine…).

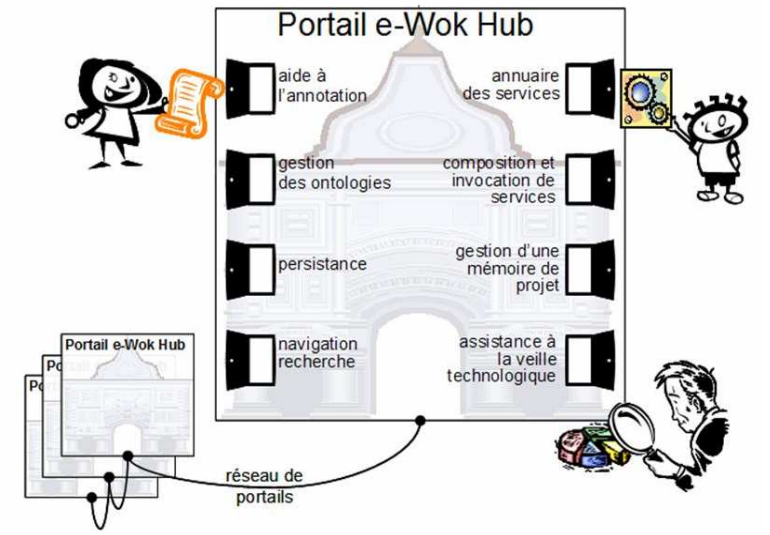


Figure 66 - Ensemble de portails web du projet E-WOK_HUB

Pour les premiers résultats, le projet E-WOK_HUB a apporté des ontologies et une base d'annotations sémantiques décrivant les connaissances et les ressources du domaine des géosciences. Nous faisons une expérimentation sur les données évolutives de ce projet, en particulier sur les deux versions de l'ontologie COG[51] (Code Officiel Géographique) de l'INSEE[52] et une trace capturant les changements entre ces deux versions ontologiques. Cette trace d'évolution est générée par l'éditeur ECCO qui est un outil d'édition de l'ontologie développé au sein du projet E-WOK_HUB.

[50] http://www.ifp.fr/

[51] http://xmlfr.org/actualites/tech/060804-0001

[52] Institut National de la Statistique et des Études Économiques

6.2.2 Expérimentations

Nous présentons dans cette section le processus d'expérimentation du système CoSWEM avec notre nouvelle approche proposée. Nous choisissons les fonctionnalités principales à expérimentées dans les deux projets ci-dessus. Ces expérimentations sont également divisées en deux contextes s'appuyant sur la disponibilité de la trace d'évolution qui capture les changements effectués entre deux versions de l'ontologie.

Première expérimentation : Avec trace d'évolution – expérimentation dans le projet E-WOK_HUB

Le projet E-WOK_HUB est lié à un scénario applicatif dans le domaine des géosciences. Ce projet vise à gérer la mémoire de plusieurs projets sur la capture et le stockage du CO2, tout en exploitant les résultats de la veille technologique sur le domaine. Par conséquent, ce scénario consiste à utiliser des ontologies décrivant le domaine des géosciences. Ces ontologies sont créées et modifiées en utilisant l'éditeur d'ontologie ECCO.

Les étapes principales de cette expérimentation sont :

- Charger dans le système CoSWEM les deux versions de l'ontologie COG et les annotations sémantiques reposant sur la première version de cette ontologie COG. Ces annotations décrivent des ressources et des documents du domaine des géosciences.

- Charger la trace d'évolution qui capture des changements effectués entre ces deux versions de l'ontologie. Cette trace est générée grâce à l'éditeur d'ontologie ECCO.

- Visualiser les changements de la trace d'évolution et aussi les éléments concernés par chaque changement ontologique (e.g. quel concept a été supprimé, quel nouveau concept fusionné à partir de deux anciens concepts existants…).

- Afficher les différences au niveau des concepts et des propriétés entre deux versions de l'ontologie COG. Pour les concepts ou les propriétés qui sont capturés dans la trace d'évolution, CoSWEM affiche aussi les informations expliquant comment ces éléments ont été modifiés.

- Chercher automatiquement les annotations sémantiques potentiellement inconsistantes qui sont affectées par les changements de la trace d'évolution. Parmi ces annotations, le système détecte les annotations sémantiques qui deviennent obsolètes et inconsistantes par rapport à la nouvelle version de l'ontologie COG. Cette étape est réalisée à l'aide des fonctions de CoSWEM combinées avec la recherche intelligente par le moteur de recherche sémantique Corese.

- Trouver automatiquement les annotations sémantiques affectées par un changement quelconque dans la trace d'évolution ainsi que les changements apparus concernant une annotation sémantique quelconque.

Deuxième expérimentation : Sans trace d'évolution – expérimentation dans le projet PALETTE

Le lot *WP3-Services de Gestion des connaissances* du projet PALETTE consiste à élaborer des ontologies décrivant les connaissances et activités des CoPs (i.e. l'ontologie O'CoP) et de développer des services de gestion des connaissances de CoPs. Ces ontologies sont normalement développées en plusieurs étapes et par différents partenaires du projet. Cela entraîne donc plusieurs versions de l'ontologie qui sont dérivées à partir d'une ontologie de base. Nous avons choisi deux versions de l'ontologie évolutive O'CoP pour notre validation dans le cas où la trace d'évolution n'est pas gardée.

L'expérimentation dans ce cas suit les étapes principales suivantes :

- Charger dans le système CoSWEM les deux versions de l'ontologie O'CoP et les annotations sémantiques reposant sur la première version de cette ontologie O'CoP. Ces annotations décrivent des ressources et des documents les Communautés de Pratique (CoPs).

- Appliquer les règles de détection d'inconsistance pour chercher automatiquement tous les éléments inconsistants (i.e. les inconsistances sur le concept, sur la propriété, sur le domaine/co-domaine, sur le type des données) de la base d'annotations par rapport à la deuxième version de l'ontologie O'CoP.

- Visualiser toutes les annotations sémantiques obsolètes et incohérentes qui contiennent ces éléments inconsistants. Pour chaque élément inconsistant, on peut aussi préciser quelles sont les annotations affectées contenant cet élément.

- Afficher les emplacements physiques correspondant aux annotations sémantiques inconsistantes ainsi que le nombre de triplets inconsistants dans chaque fichier d'annotation.

- Proposer des solutions permettant à l'utilisateur de choisir une stratégie convenable afin de corriger une annotation sémantique inconsistante d'une manière semi-automatique. Pour les annotations qui concernent le même élément inconsistant (e.g. celles qui contiennent le concept obsolète c), il est possible d'appliquer une même solution pour traiter en même temps toutes ces annotations.

- Changer et mettre à jour certains paramètres du système CoSWEM tels que les chemins des répertoires de l'ontologie, de l'ancienne/nouvelle

base d'annotations, de règles… Ces paramètres guident le fonctionnement du CoSWEM selon les exigences de l'utilisateur.

6.3 Évaluation des résultats

6.3.1 Exigences des fonctionnalités

Nous présentons par la suite certains exigences générales des fonctionnalités qui doivent être satisfaites pour évaluer le fonctionnement du système CoSWEM.

- ***Fonctionnalités principales :*** Le système doit fournir des interfaces (ou des menus) qui permettent à l'utilisateur d'effectuer les opérations générales suivantes :
  - Charger des versions de l'ontologie évolutive et des annotations sémantiques basées sur cette ontologie.
  - Charger la trace d'évolution dans le scénario d'évolution où on peut garder cette trace. Il y a aussi l'interface pour représenter cette trace d'évolution avec ses changements effectués capturés.
  - Changer des paramètres tels que les répertoires contenant l'ontologie, la base d'annotations sémantiques, les règles…
  - Appliquer des tâches de détection et de correction d'inconsistance.
  - Recharger le système CoSWEM en cas de modification des paramètres.
- ***Comparaison des différences entre deux versions de l'ontologie*** : Pour cette tâche, le système doit afficher clairement les différences entre deux versions de l'ontologie selon les cas suivants :
  - S'il n'y pas de trace de changements, CoSWEM doit lister d'une manière simple les concepts et les propriétés qui diffèrent entre deux versions de l'ontologie : par exemple, les concepts qui existent dans la première version mais n'existent pas dans la deuxième version et vice-versa.
  - Dans le cas où l'on possède l'information sur les changements effectués dans la trace d'évolution, CoSWEM doit fournir plus d'informations décrivant la différence entre deux versions ontologiques à propos de chaque changement effectué. Par exemple, pour la disparition d'un concept c, la raison de cette disparition doit être explicitée et visualisée : une suppression, un

renommage ou une fusion de concept… Cette information a été capturée dans la trace d'évolution.

- D'autre part, toutes les informations décrivant les différences entre deux versions de l'ontologie doivent être affichées clairement et d'une manière compréhensible (e.g. les éléments différents portent différentes couleurs…)

- ***Détection d'inconsistance :*** La fonction de détection d'inconsistance de CoSWEM doit permettre de:

  - Chercher des éléments inconsistants sur chaque type (i.e. concept, propriété, domaine, co-domaine, type des données) et chercher des annotations sémantiques inconsistantes à cause des changements de l'ontologie.

  - Lister toutes les annotations inconsistantes qui sont causées par un changement, et inversement afficher tous les changements effectués se rapportant à un fichier d'annotation.

- ***Résolution d'inconsistance :*** Pour la correction des inconsistances, le système doit fournir des interfaces permettant de :

  - Choisir une solution convenable et des éléments à remplacer pour corriger l'annotation inconsistante.

  - Les annotations corrigées doivent garder le même format que celui d'avant la modification.

6.3.2 Mesures d'évaluation

Nous avons indiqué dans l'état de l'art et dans le chapitre 2 qu'il n'existe pas actuellement de systèmes ou d'outils ayant des fonctions similaires à CoSWEM au niveau de la détection et de la résolution des annotations sémantiques inconsistantes. Il n'est donc pas possible de faire des évaluations comparatives des résultats acquis avec d'autres outils afin de comparer et évaluer la performance des fonctions similaires de notre système.

Cependant, pour la phase de détection d'inconsistance, nous avons besoin d'évaluer plus concrètement son efficacité à travers des résultats détectés. Nous établissons donc des critères concrets (i.e. la précision, le rappel et la F-mesure) dans cette phase. Dans le domaine de l'extraction d'information, nous avons utilisé deux mesures fréquemment utilisées dans l'évaluation des systèmes d'extraction d'informations, à savoir la précision et le rappel :

- *La précision (P)* est le pourcentage des termes correctement extraits ; cela mesure donc l'absence de bruit dans l'extraction.

$$Précision = \frac{Nombre\ de\ termes\ correctement\ extraits}{Nombre\ total\ des\ termes\ extraits}$$

- *Le rappel (R)* est le pourcentage des termes correctement extraits par rapport aux termes qui auraient dû être extraits ; cela mesure l'absence de silence dans l'extraction.

$$Rappel = \frac{Nombre\ de\ termes\ correctement\ extraits}{Nombre\ des\ termes\ qui\ auraient\ dû\ être\ extraits}$$

Dans notre évaluation, nous avons besoin de connaître le nombre de triplets (ou d'annotations) inconsistants renvoyés par CoSWEM après la phase de détection d'inconsistance. En plus, nous nous intéressons aussi au ratio de triplets inconsistants détectés qui sont évalués correctement. Par conséquent, nous donnons par la suite notre aperçu sur ces deux critères :

- *La précision (P)* est le pourcentage des triplets inconsistants correctement détectés sur le nombre total des triplets inconsistants détectés.

$$Précision = \frac{Nombre\ de\ triplets\ inconsistants\ correctement\ détectés}{Nombre\ total\ de\ triplets\ inconsistants\ détectés}$$

- *Le rappel (R)* est le pourcentage des triplets inconsistants correctement détectés par rapport aux triplets inconsistants existants dans la base d'annotations.

$$Rappel = \frac{Nombre\ de\ triplets\ inconsistants\ correctement\ détectés}{Nombre\ total\ de\ triplets\ inconsistants\ existants}$$

La Figure 67 représente concrètement les mesures utilisées pour évaluer les résultats détectés par CoSWEM. Nous utilisons des termes *Vrais Positifs (VP), Vrais Négatifs (VN), Faux Positifs (FP)* et *Faux Négatifs (FN)* afin d'exprimer le pourcentage des triplets inconsistants détectés en utilisant CoSWEM par rapport aux triplets existants dans la base d'annotations. Les taux de Précision et de Rappel peuvent être aussi calculés par ces termes comme suit:

$$Précision = \frac{VP}{VP + FP} \qquad Rappel = \frac{VP}{VP + FN}$$

D'autre part, nous utilisons la *F-mesure (F)* qui est un compromis entre le rappel et la précision (dite aussi moyenne harmonique du rappel et de la précision). Elle permet d'évaluer la performance de la détection des triplets inconsistants par une seule mesure. La F-mesure est définie par :

$$F\text{-}mesure\ (F) = \frac{2*Précision*Rappel}{Précision + Rappel}$$

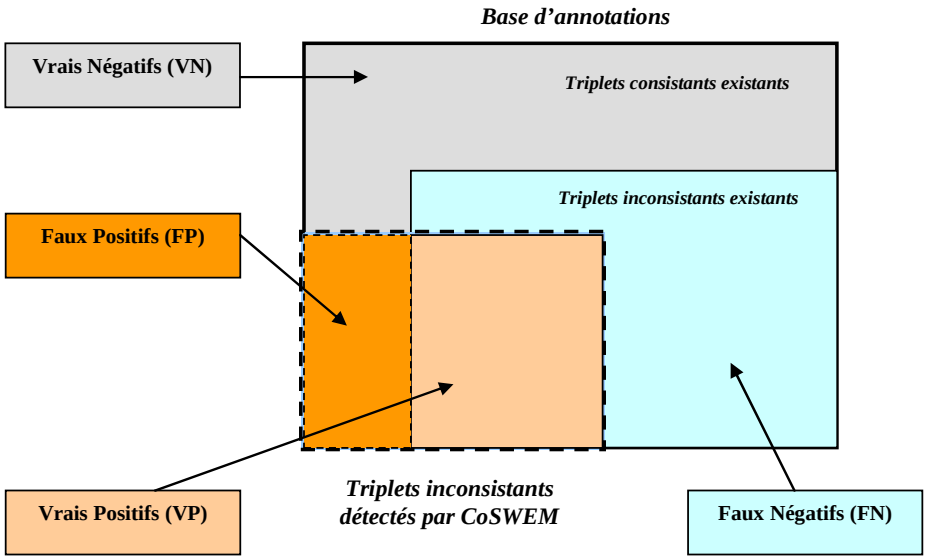


Figure 67 – Mesures utilisées pour évaluer les résultats détectés

Nous notons que :

- **D** : le nombre total des triplets inconsistants détectés renvoyés par CoSWEM.
- **E** : le nombre total des triplets inconsistants existants dans la base d'annotations. Il est déterminé manuellement par l'ingénieur du sysème (dans notre validation, nous connaissons les triplets inconsistants existants par rapport la nouvelle version de l'ontologie).
- **C** : le nombre de triplets inconsistants correctement détectés par CoSWEM (Vrais Positifs - VP)
- **FP = D - C** : le nombre des triplets inconsistants incorrects détectés (Faux Positifs - FP)
- **FN = E - C** : le nombre des triplets vraiment inconsistants mais pas détectés (Faux Négatifs - FN).
- Ainsi, **Précision = C / D ; Rappel = C / E ; F-mesure = 2*P*R/(P+R).**

6.3.3 Évaluation des résultats

Pour la première expérimentation sur le projet E-WOK_HUB, nous faisons des tests avec deux versions de l'ontologie COG. Chaque version possède environ 44 concepts et une dizaine de propriétés. La COG décrit les connaissances et les ressources du domaine de géosciences et particulièrement de la géographie. Elle comporte certains concepts principaux tels que `Groupement_de_pays`,

`Pays_ou_Territoire`, `Pays`, `Commune`, `Departement`... pour désigner les niveaux de régions, les concepts `Type_administratif`, `Type_localite`... pour décrire les caractéristiques de région, de localité, etc. Ces concepts sont reliés par certaines propriétés *`code_canton`*, *`code_commune`*... pour indiquer les codes de chaque niveau de région, les régions qui sont adjacentes (i.e. la propriété *`voisin`*), la population de chaque région (i.e. la propriété *`population`*), etc. La Figure 68 représente partiellement un extrait des hiérarchies de concepts et de propriétés de cette ontologie COG.

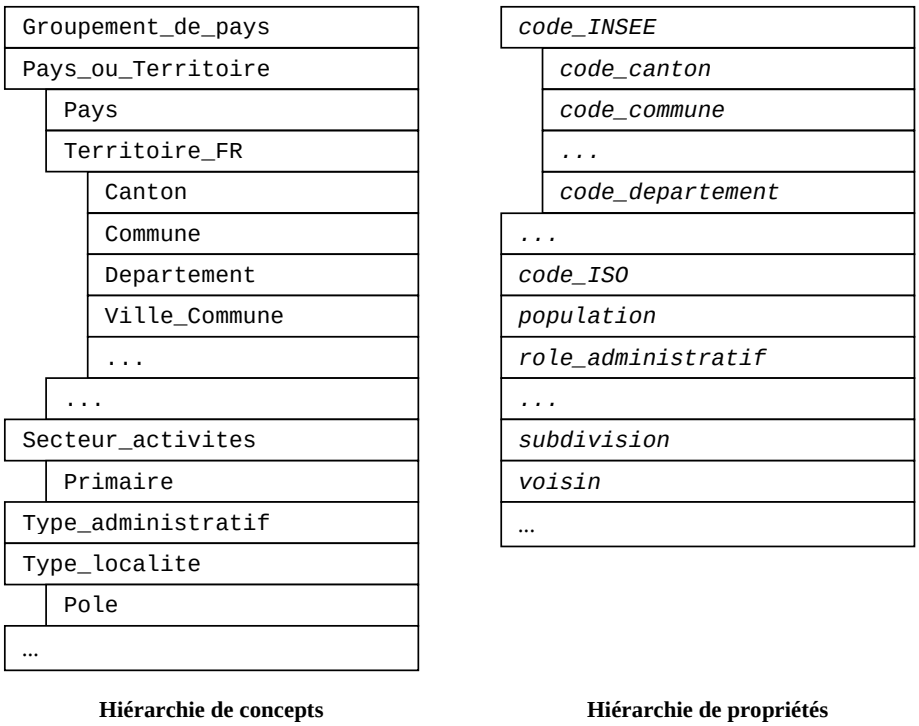


Figure 68 - Certains concepts et propriétés principaux de l'ontologie COG

Dans la base d'annotations expérimentée, nous avons 11 fichiers d'annotations sémantiques (avec plus de 600 triplets, chaque fichier d'annotation contient plusieurs triplets) qui reposent sur l'ancienne version de l'ontologie COG. Ces fichiers d'annotations sémantiques ont été créés et utilisés par les utilisateurs et les experts du projet E_WOK_HUB. Il y a également une trace d'évolution qui capture les changements effectués entre ces deux versions COG (c.f. Annexe C). Cette trace est représentée sous forme d'une annotation sémantique reposant sur les termes définis de l'ontologie d'évolution ainsi que les concepts et les propriétés modifiés de l'ontologie domaine COG. Elle a capturé quelques changements effectués (i.e. `DeleteConcept`, `CreateConceptHierarchyLink`, `RenameConcept`...) qui affectent sur les concepts de l'ontologie COG tels que `Canton`, `Ville_Commune`...

Tableau 12- Résultat de la détection des annotations inconsistantes

Changement effectué	Eléments affectés	Nombre d'annotations inconsistantes	Nombre d'annotations inconsistantes détectées
`DeleteConcept`	`Ville_Commune`	3	3
`CreateHierarchy-ConceptLink`	`Ville, Territoire_FR`	0	0
`RenameConcept`	`Canton, «Canton-Ville»`	6	6
`InsertConcept`	`Capital`	0	0
`CreateProperty-DomainLink`	`Capital, role_administratif`	0	0
`RemoveConcept-DomainLink`	`Ville, role_administratif_ville`	2	2

En exploitant les informations de cette trace d'évolution, nous avons intégré dans CoSWEM le mécanisme de recherche sémantique de Corese avec les tags dédiés de SemTag API (e.g. tag qui est dédié à la requête de recherche d'un triplet concret, tag qui permet de chercher des annotations contenant un type de concept quelconque…) pour chercher des inconsistances des annotations sémantiques générées par les changements de l'ontologie. Le résultat de la détection des fichiers d'annotations inconsistantes est très pertinent et exact. Parmi les 11 fichiers d'annotations (chaque fichier comporte une annotation avec plusieurs triplets), CoSWEM a trouvé 9 fichiers d'annotations potentiellement inconsistantes (i.e. les annotations qui contiennent les éléments affectés par les changements effectués de l'ontologie). Nous trouvons ensuite 3 changements de correction facultative (i.e. `CreateHierarchyConceptLink`, `InsertConcept`, `CreatePropertyDomainLink`) (c.f. premier colonne, Tableau 12) grâce à la propriété `<hasAnnotInconsistency>` qui n'entraînent pas des inconsistances sur l'annotation sémantique. Concernant ces changements de correction facultative, aucun fichier d'annotation inconsistante n'est détecté (i.e. lignes avec la valeur 0 dans le Tableau 12). Après avoir déterminé les annotations ne contenant que les changements de correction facultative, c-à-d les annotations restant encore consistantes dans l'ensemble des annotations potentiellement inconsistantes, nous avons obtenu 8 fichiers d'annotations qui sont vraiment inconsistantes. Le Tableau 12 montre le nombre de fichiers d'annotations inconsistantes (i.e. la 3ème colonne) contenant l'élément affecté équivalent (i.e. la 2ème colonne). La quatrième colonne de ce tableau représente le nombre de

fichiers d'annotations inconsistantes détectées par CoSWEM. Nous pouvons voir concrètement les fichiers d'annotation qui contiennent les éléments affectés par les changements effectués de l'ontologie. Par exemple, la Figure 69 montre les fichiers d'annotations sémantiques trouvés (i.e. 6 fichiers d'annotations) : ces annotations étant devenues inconsistantes car elles contiennent le concept renommé `Canton` (i.e. il existe le changement effectué `RenameConcept(Canton, "Canton-Ville")` dans la trace d'évolution), ce concept Canton n'existe plus dans la nouvelle version de COG. De manière similaire, nous trouvons d'autres fichiers d'annotations inconsistantes à cause d'autres changement effectués.

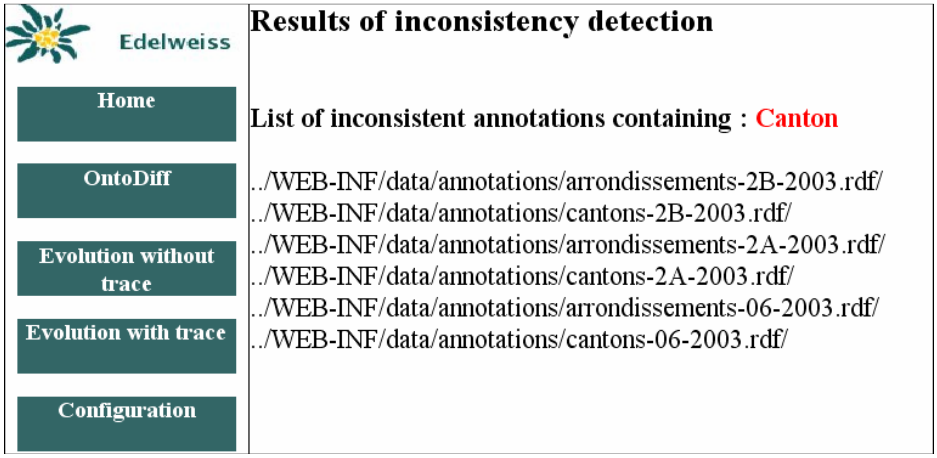


Figure 69 - Liste des annotations inconsistantes contenant le concept renommé `Canton`

Dans la deuxième expérimentation sur le projet PALETTE, nous avons deux versions évolutives de l'ontologie O'CoP. Chaque version a environ 240 concepts et 80 propriétés. Cette ontologie décrit les connaissances du domaine et des services pour les CoPs telles que les types d'activités de CoPs (i.e. les concepts `ACTIVITY`, `Individual_Activity`, `Collaborative_Activity`...), les caractéristiques de CoPs (i.e. les concepts `CoP_s_characteristics`, `Diversity`...), les acteurs impliqués dans un CoP (i.e. les concepts `actors`, `personnes_physiques`...) ou les rôles existants dans un CoP (i.e. les concepts `role_in_the_CoP`, `Governance_role`, `Member_role`...), etc. La Figure 70 présente certains concepts principaux de cette ontologie O'CoP ainsi qu'un extrait de la hiérarchie de propriétés. Les changements effectués entre ces deux versions de l'ontologie O'CoP sont environ 121 changements sur les concepts et 22 changements sur les propriétés. Ces changements sont de plusieurs types différents tels que la suppression du concept et de la propriété, le renommage du concept, la suppression du lien entre un concept et une propriété, etc.

Pour la base d'annotations reposant sur l'ancienne version de O'CoP, nous avons créé environ 10 fichiers d'annotations sémantiques (avec 85 triplets approximativement) dans lesquels il y a des annotations consistantes ainsi que des annotations inconsistantes (par rapport la nouvelle version de O'CoP) en vue de tester notre approche de détection d'inconsistance basée sur des règles. Chaque

fichier contient une annotation sémantique comportant plusieurs triplets et les instances des concepts de l'ontologie O'CoP. Parmi les annotations inconsistantes, il existe différents triplets contenant tous les types d'inconsistance possibles que nous avons présentés dans le jeu de tests du chapitre 5 (c.f. section 5.3). Ces inconsistances sont distribuées aléatoirement dans les fichers d'annotations sémantiques. Au cours de l'évolution, la trace de changements entre ces deux versions de l'ontologie O'CoP n'a pas été conservée.

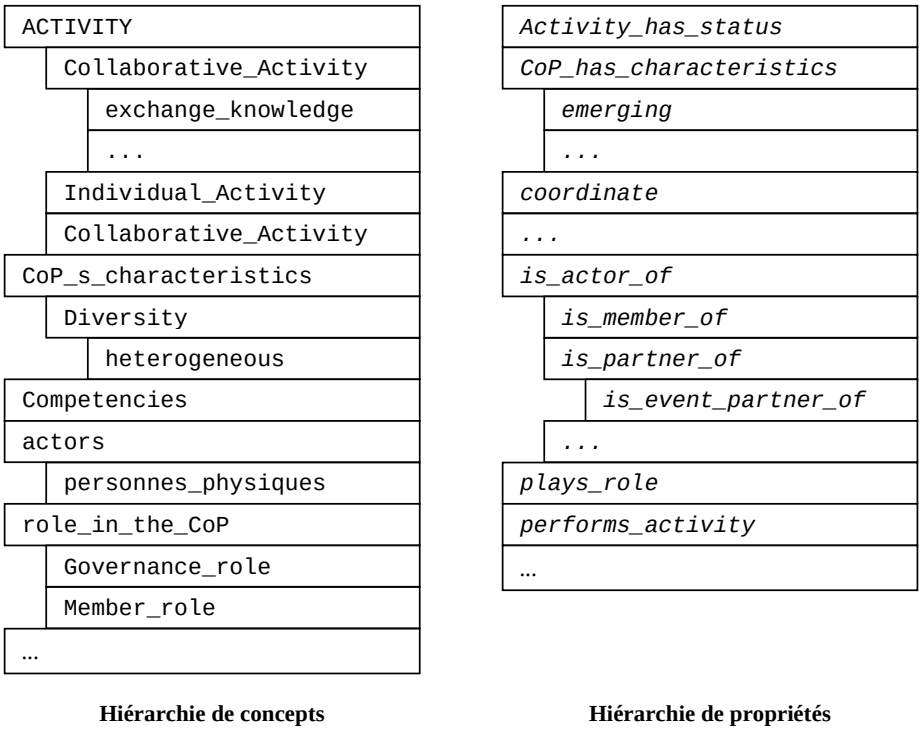


Figure 70 - Certains concepts et propriétés principaux de l'ontologie O'CoP

En appliquant les règles de détection d'inconsistance, le système CoSWEM renvoie tous les éléments (i.e. le concept, la propriété, le domaine/co-domaine, le type des données) de la base d'annotations qui deviennent obsolètes et inconsistants par rapport à la nouvelle version de l'ontologie O'CoP. Nous présentons par la suite certaines interfaces qui affichent les résultats d'inconsistance détectés.

La Figure 71 montre une liste des concepts inconsistants détectés dans la base d'annotations. Ces concepts ne sont pas conformes à la structure de la nouvelle version de O'CoP (e.g. les concepts `actors1`, `actors2`… n'existent plus dans la nouvelle version de l'ontologie). Dans cette interface, nous pouvons voir aussi le nombre de triplets ainsi que le nombre de fichiers d'annotations contenant l'élément inconsistant. Par exemple si on s'intéresse au concept inconsistant `actors1`, il est présent dans un triplet et dans un fichier d'annotation sémantique. D'autre part, nous connaissons également les fichiers d'annotations affectés via leur emplacement physique, par exemple le concept inconsistant `personnes_physiques3` est présent dans deux fichiers d'annotations qui ont pour emplacements respectifs : *C:/Program Files/Apache Software Foundation/Tomcat 5.5/webapps/coswem-ifi/WEB-INF/coswem-withouttrace/data/annotations-/annot2.rdf* et *C:/Program Files/Apache Software Foundation/Tomcat 5.5/webapps/coswem-ifi/WEB-INF/coswem-withouttrace/data/annotations-/annot5.rdf.* En accédant aux liens de solutions (i.e. la colonne à droite - Solution), le système affichera les solutions possibles pour corriger les annotations inconsistantes détectées ci-dessus.

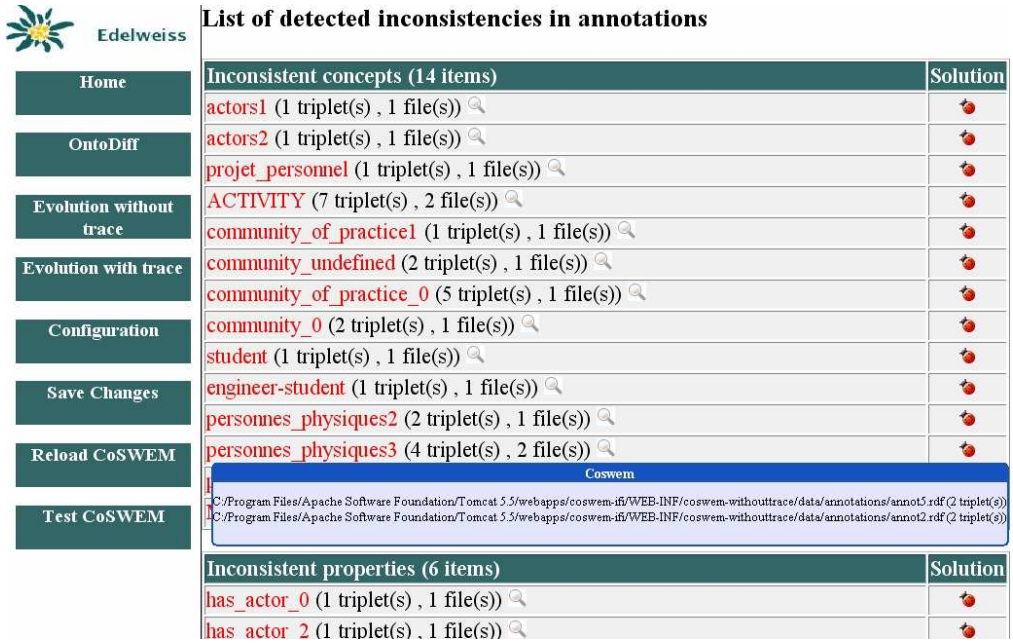


Figure 71 - Liste des concepts inconsistants détectés

De même, le résultat des propriétés inconsistantes détectées est montré dans la Figure 72. La ligne au début de cette figure indique le nombre de propriétés détectées (e.g. 5 items) dans la base d'annotations qui sont inconsistantes par rapport à la nouvelle version de l'ontologie O'CoP. La colonne à droite (c.f. Figure 72) contient des liens (i.e. View solutions) pour afficher des solutions afin de résoudre ces propriétés inconsistantes.

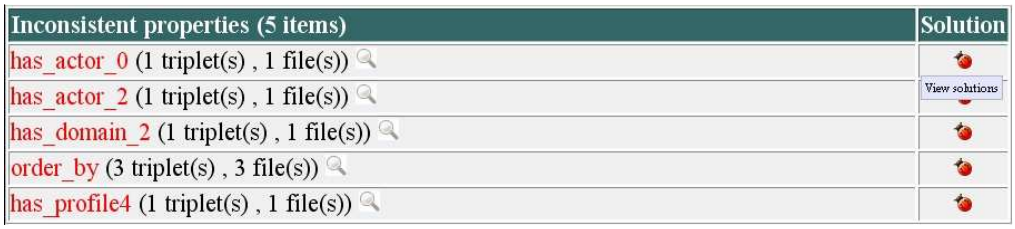


Figure 72 - Liste des propriétés inconsistantes détectées

Nous présentons une autre interface représentant la liste des inconsistances détectées sur le domaine (c.f. Figure 73). Cette figure indique les paires constituées d'un concept domaine et d'une propriété reliés dans la base d'annotations qui sont obsolètes et inconsistantes par rapport à la nouvelle version de l'ontologie. Par exemple dans une annotation sémantique, la propriété *`coordinate`* porte sur le concept `Facilitator` (i.e. la deuxième colonne) mais dans la nouvelle version de l'O'CoP, elle a pour le concept domaine `Coordinator` (i.e. la troisième colonne). Cela est détecté comme une inconsistance sur le domaine. Dans cette interface, on peut toujours obtenir les fichiers d'annotation contenant des inconsistances sur le domaine ainsi qu'accéder aux liens de solutions pour corriger ces inconsistances.

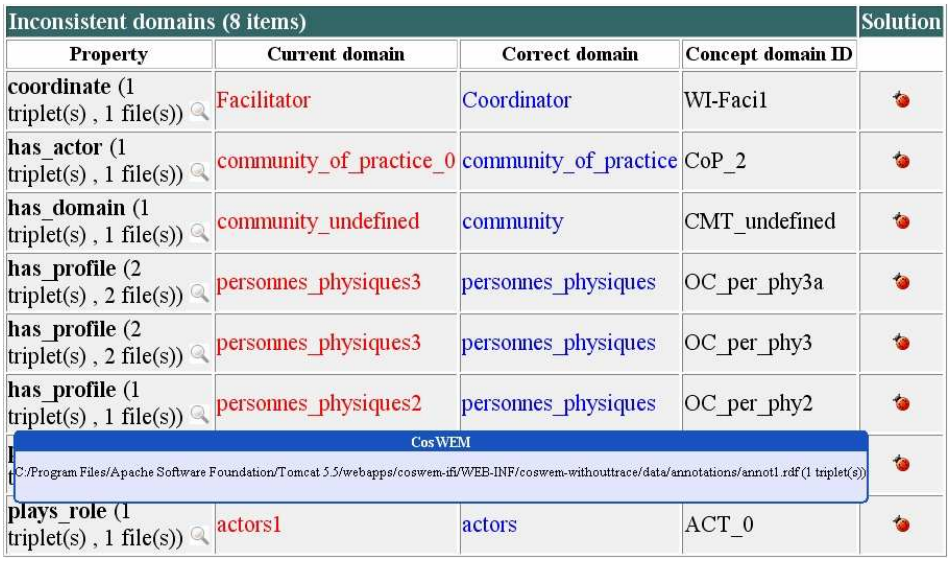


Figure 73 - Liste des inconsistances de domaine

Notons que si un triplet contient un concept inconsistant qui appartient au domaine d'une propriété correcte dans ce triplet, alors ce triplet est détecté dans les deux positions : inconsistance sur le concept et inconsistance sur le domaine. Par exemple, supposons qu'il existe une relation entre le concept `personnes_physiques3` et la propriété *`has_profile`* dans l'annotation, mais que le concept `personnes_physiques3` ne soit pas défini dans la nouvelle version de l'ontologie. Le système CoSWEM détecte alors deux inconsistances dans ce cas : le concept inconsistant `personnes_physiques3` et le domaine

inconsistant entre `personnes_physiques3` et la propriété *`has_profile`*. Cette manière de détection d'inconsistance est aussi appliquée pour chercher des inconsistances sur le concept et sur le co-domaine en même temps.

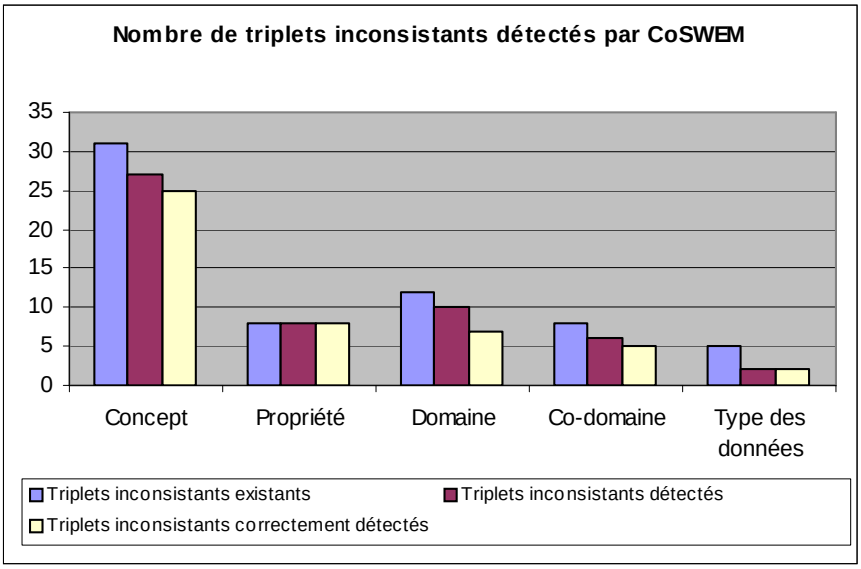


Figure 74 – Diagramme de nombre des triplets inconsistants détectés

Le Tableau 13 présente les résultats de la phase de détection d'inconsistance sur la base d'annotations. Nous nous intéressons aux trois facteurs : (i) nombre de triplets inconsistants existants dans la base d'annotations sémantiques par rapport à la nouvelle version de l'ontologie O'CoP, (ii) nombre de triplets inconsistants détectés par les règles de détection de CoSWEM et (iii) nombre de triplets inconsistants correctement détectés (selon le besoin de l'utilisateur). Les résultats sur le nombre de triplets inconsistants détectés sont également représentés dans un diagramme graphique (c.f. Figure 74).

Tableau 13 - Nombre de triplets inconsistants détectés sur chaque élément

Triplets inconsistants sur	**Nr. de triplets inconsistants existants (E)**	**Nr. de triplets inconsistants détectés (D)**	**Nr. de triplets inconsistants correctement détectés (C)**
Concept	31	27	25
Propriété	8	8	8
Domaine	12	10	7
Co-domaine	8	6	5
Type des données	5	2	2
Total	**64**	**53**	**47**

Afin d'évaluer le résultat de détection sur chaque élément (i.e. le concept, la propriété, le domaine…), nous calculons tout d'abord les proportions des nombres de triplets renvoyés qui affectent les mesures de validation telles que VP, FP et FN (c.f. Figure 67). En se basant sur ces proportions, nous calculons ensuite les mesures de précision et de rappel ainsi que de F-mesure sur le nombre de triplets inconsistants détectés correspondant à chaque élément (c.f. Tableau 14). En regardant ce tableau, nous aurons une vue d'ensemble sur la proportion des triplets inconsistants qui sont détectés sur chaque type d'élément. Par exemple la quatrième ligne (c.f. Tableau 14) montre que les triplets correctement détectés contenant les concepts inconsistants couvrent 93%, les triplets inconsistants contenant les propriétés inconsistantes sont entièrement détectés avec une précision de 100%, les triplets correctement détectés qui contiennent les domaines inconsistants couvrent 70% de ceux qui sont existants (i.e. la cinquième ligne), etc. La Figure 75 représente également une vue globale des trois mesures (i.e. la précision, le rappel et la F-mesure) sous forme d'un diagramme graphique. Nous analysons les raisons affectant ces résultats dans la section suivante.

	Concept	**Propriété**	**Domaine**	**Co-domaine**	**Type des données**
Faux Positifs (FP)	2	0	3	1	0
Faux Négatifs (FN)	6	0	5	3	3
Précision (P)	0,93	1,00	0,70	0,83	1,00
Rappel (R)	0,81	1,00	0,58	0,63	0,40
F-mesure (F)	0,86	1,00	0,64	0,71	0,57

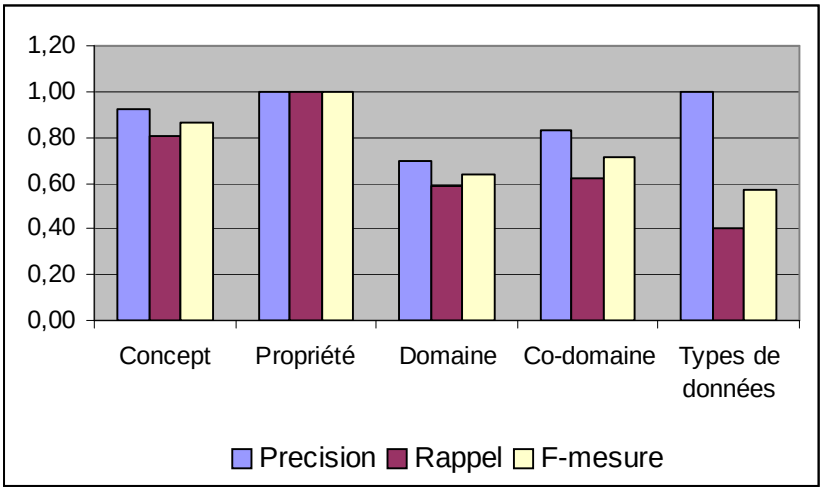


Figure 75 – Diagramme du résultat des mesures P, R et F-mesure

Après avoir détecté les triplets inconsistants sur chaque type élément, ceux-ci ont besoin d'être corrigés. La Figure 76 montre une interface qui fournit à l'utilisateur des solutions pour choisir comment corriger ces inconsistances

existantes dans les fichiers d'annotation. D'une manière simple, on peut choisir de supprimer un triplet contenant l'élément inconsistant. La fonction d'édition de CoSWEM enlèvera temporairement ce triplet du fichier d'annotation sémantique qui le contient. D'autre part, l'utilisateur peut choisir un élément correct de la nouvelle version de l'ontologie O'CoP afin de remplacer celui qui crée l'inconsistance dans le triplet. Dans ce cas, un algorithme a été implémenté dans CoSWEM pour chercher les "meilleurs" éléments à remplacer. Par exemple, dans l'interface de solutions pour corriger le concept inconsistant `personnes_physiques3` (i.e. Figure 76), les concepts "les plus proches" (i.e. les concepts `personnes_physiques`, `Teacher`, etc.) de `personnes_physiques3` dans la nouvelle version de O'CoP sont classés au début de la liste des concepts proposés à remplacer. Sinon, on peut toujours chercher un autre élément disponible dans l'ontologie grâce à un menu *drop-down* (c.f. Figure 76) qui contient le reste des éléments de la nouvelle version de l'ontologie. Cela assure que le triplet (ou l'annotation) après sa correction sera conforme à la nouvelle ontologie.

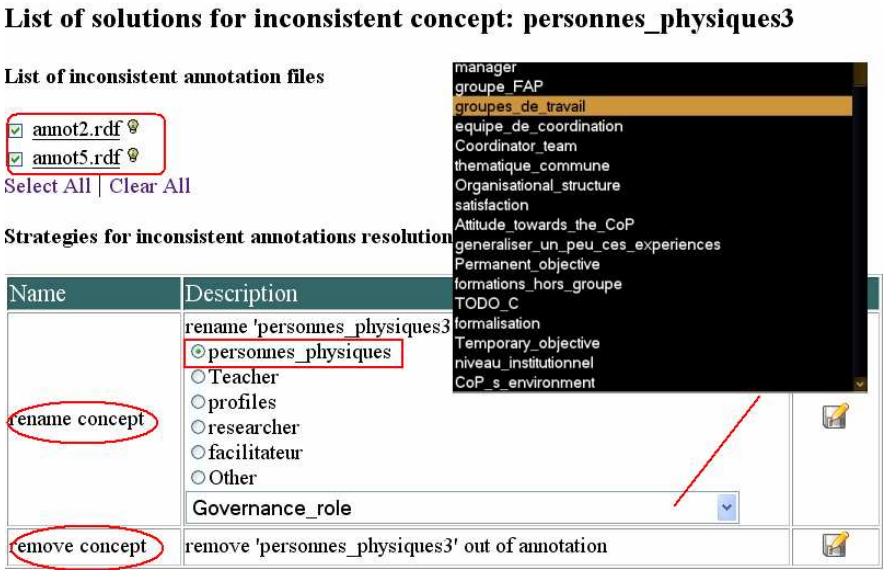


Figure 76 - Interface de correction des triplets inconsistants

6.3.4 Discussion

En général, les expérimentations ci-dessus ont montré que le système CoSWEM peut répondre à des exigences de fonctionnalités sur certains aspects principaux tels que les interfaces homme-machine, les menus, les fonctions dédiées à la détection et à la correction d'inconsistance, la comparaison des différences des versions de l'ontologie, etc.

Dans la première évaluation sur le projet E-WOK_HUB, les informations de changement sont capturées dans une trace d'évolution. Le système CoSWEM a

profité du mécanisme de recherche sémantique de Corese pour exploiter cette trace. En analysant ces informations, il peut connaître non seulement les types de changements effectués mais aussi les éléments affectés par ces changements. Par exemple avec une information capturée dans la trace : `<csw:DeleteConcept>…<csw:hasDeleteConcept rdf:resource="…#Ville_Commune"/>… </csw:DeleteConcept>`, il sait que le concept `Ville_Commune` a été supprimé (i.e. `DeleteConcept`). CoSWEM peut détecter ensuite les annotations sémantiques ou les triplets contenant ce concept supprimé. Les résultats détectés dans ce cas sont satisfaisants et précis grâce à la recherche exacte à l'aide d'une requête dans la base d'annotations ainsi que dans les ontologies. Du point de vue technique, nous avons implémenté dans CoSWEM différentes "instances" de Corese pour réaliser des tâches spécifiques. Chacune est dédiée à la recherche d'information sur une base de connaissances concrète (e.g. une instance Corese dédiée à l'exploitation de la base d'annotations, une autre dédiée à faire des requêtes sur la nouvelle version de l'ontologie, etc.).

Pour la deuxième évaluation dans le projet PALETTE, les résultats détectés sont aussi satisfaisants. Le moteur de recherche facilite également le processus de recherche des éléments dans la base d'annotations et dans l'ontologie avec une bonne précision. Le Tableau 14 a montré une proportion élevée de triplets détectés qui sont correctement inconsistants (i.e. le rappel de la détection d'inconsistance sur le concept est 81% et sur la propriété est 100%, etc.). Les valeurs de la F-mesure montrent aussi que nous avons un bon compromis entre le rappel et la précision. Cependant, la trace d'évolution n'est pas gardée, il nous manque donc les informations nécessaires pour détecter toutes les inconsistances existantes. Il y a encore quelques obstacles au niveau technique qui affectent l'efficacité des résultats détectés :

- Pour la détection d'inconsistance sur le concept, s'il existe en même temps dans l'annotation une instance incohérente et un triplet contenant un même type de concept inconsistant, CoSWEM ne les distingue pas et il calcule comme s'il s'agissait d'une seule inconsistance (en effet, il serait calculé en deux inconsistances). Par exemple pour une instance du type de concept inconsistant `actors1 <oc:actors1 rdf:about="…#ACT_1"/>` et un triplet contenant le type de concept `<oc:actors1 rdf:about="…#ACT_1"><oc:plays_role rdf:resource="…#roleCoP1"/></oc:actors1>`, le système considère comme une inconsistance détectée sur le concept `actors1`. Cela explique que le nombre de triplets détectés soit moins grand que le nombre de ceux existant dans la base d'annotations.

- Pour la détection d'inconsistance sur le domaine, s'il existe en même temps dans un triplet le concept domaine inconsistant et la propriété inconsistante (i.e. ce concept et cette propriété ne sont pas définis dans l'ontologie), CoSWEM ne compte pas cette inconsistance. La raison de cet obstacle est que CoSWEM effectue des règles de détection

d'inconsistance sur la propriété avant celles sur le domaine, donc si l'inconsistance a déjà été détectée sur la propriété, elle ne sera pas prise en compte pour l'inconsistance sur domaine. Par exemple pour le triplet `<oc:actors1 rdf:about="…#ACT_1"> <oc:plays_role1 rdf:resource="…# roleCoP1"/></oc:actors1>` dans lequel le concept `actors1` et la propriété *`plays_role1`* n'existent pas dans l'ontologie, CoSWEM considère que ce triplet est inconsistant sur la propriété mais pas sur le domaine. Cependant, si la propriété est bonne et le concept domaine est inconsistant dans un triplet, le système peut détecter cette inconsistance et il la considère alors comme une inconsistance.

- De même, CoSWEM ne prend pas en compte l'inconsistance sur le co-domaine dans un triplet s'il existe en même temps le concept co-domaine inconsistant et la propriété inconsistante.

- D'autre part, si un triplet contient plusieurs éléments inconsistants, il est considéré plusieurs fois comme inconsistant, mais en réalité il est considéré comme étant un triplet inconsistant unique. Cela explique que le nombre de triplets détectés correctement est moins grand que le nombre de triplets détectés par CoSWEM.

Cependant, nos expérimentations ont également montré quelques limites :

- Dans l'expérimentation sur le projet E-WOK_HUB, les concepts et les propriétés des versions de l'ontologie COG sont peu nombreux. En plus, les changements de l'ontologie effectués ne sont pas très variés et ils contiennent principalement des changements simples (i.e. `RenameConcept`, `InsertConcept`, `CreatePropertyDomainLink`…). Par conséquent, la complexité de cette expérimentation n'est pas très élevée.

- Dans un premiers temps du projet PALLETTE, les versions d'ontologies évolutives O'CoP utilisées étaient en cours d'amélioration par les partenaires du projet. Ces versions de cette l'ontologie et les changements ontologiques effectués sont utiles pour notre expérimentation. Cependant, les partenaires n'avaient pas encore créé une base d'annotations reposant sur l'ontologie O'CoP pour décrire leur ressources du domaine de CoPs. Par conséquent, nous avons dû établi un ensemble des annotations sémantiques en vue de tester notre approche de détection et de correction des annotations inconsistantes à cause de la modification de l'ontologie de référence. Bien que cette base d'annotations créée couvre tous les cas d'inconsistance possibles que nous avons proposées dans le jeu de tests (c.f. section 5.3) et qu'elle marche également bien avec notre expérimentation, nous avons encore besoin d'expérimenter notre approche avec les annotations sémantiques inconsistantes réelles fournies par les partenaires du projets.

- D'autre part, nous n'avons pas encore expérimenté la correction des annotations inconsistantes d'une manière automatique. Nous manquons d'informations sur les stratégies de résolution qui ont été appliquées sur l'ontologie, il est donc impossible d'effectuer une stratégie de résolution équivalente pour l'annotation sémantique afin de corriger cette annotation inconsistante.

Une des perspectives pour le système CoSWEM dans l'expérimentation et l'évaluation serait de raffiner le résultat de détection d'inconsistance. Nous ne disposons pas encore de techniques pour bien calculer le nombre des inconsistances détectées dans le cas où elles appartiennent à un triplet unique (e.g. des inconsistances à la fois sur le concept domaine, la propriété, le concept co-domaine…). D'autre part, nous aurons ensuite besoin d'un ensemble de tests avec les données réelles et évolutives (fournies par les projets du domaine) qui soient assez nombreuses et complexes possibles pour expérimenter notre approche. Cela assurera l'objectivité de notre expérimentation.

6.4 Conclusion

Dans ce chapitre, nous avons présenté l'implémentation et l'exploitation du système CoSWEM dans des scénarios réels. Ce système est implémenté en se basant sur l'architecture MVC (Modèle-Vue-Contrôleur) qui est un modèle de conception permettant de séparer le modèle de données, l'interface utilisateur et la logique de contrôle. Nous décrivons aussi la conception des composants principaux et des règles intégrées dans ce système. L'application de CoSWEM est développée en langage Java, JSP/Servlet et d'autres langages de représentation tel que HTML, XML… elle est déployée dans l'environnement du Web en utilisant le serveur Web Tomcat Apache.

Dans un premier temps, le système CoSWEM a été expérimenté dans le cadre des projets PALETTE et E-WOK_HUB avec un ensemble de données réelles et évolutives provenant de ces projets. L'expérimentation a été réalisée dans deux scénarios d'évolution qui correspondent à ce qui se passe souvent réellement : (i) avec trace d'évolution et (ii) sans trace d'évolution de l'ontologie.

Afin d'évaluer les résultats d'expérimentation du système et la qualité de la détection d'inconsistance sur les annotations sémantiques, nous reposons sur certaines mesures telles que les Vrais Positifs, Faux Positifs, Faux Négatifs… et particulièrement sur les taux importants, i.e. la précision, le rappel et la F-mesure. Nous avons présenté des interfaces montrant les résultats des inconsistances détectées pour chaque type d'élément (i.e. le concept, la propriété, le domaine…) et nous avons discuté finalement sur ces résultats en soulignant les points positifs et les limites de cette expérimentation.

Conclusion

Dans cette partie, nous faisons d'abord un résumé des travaux présentés dans ce manuscrit en soulignant les innovations apportées par cette thèse. Ensuite, nous exposons la problématique énoncé dans l'introduction de ce mémoire, nous passons en revue également les questions de recherche demandées ainsi que les contributions dans le cadre de cette thèse. Finalement, nous discutons les limites de notre approche et présentons les travaux possibles dans le futur.

Résumé de thèse

Dans ce manuscrit, nous avons présenté une nouvelle approche de la ***gestion de l'évolution du Web sémantique d'entreprise (WSE)***. C'est une approche de la gestion des connaissances pour la prochaine génération du Web sémantique dans laquelle on donne à une information un sens bien défini pour permettre aux ordinateurs et aux utilisateurs de travailler en coopération. Nous nous sommes focalisés particulièrement sur l'évolution de l'ontologie et de l'annotation sémantique qui sont deux composants importants du WSE. L'annotation sémantique à partir d'ontologies semble actuellement l'approche la plus prometteuse pour partager et exploiter l'information sur le Web. Nous nous sommes intéressés au contexte d'évolution dans lequel les changements de l'ontologie peuvent affecter la consistance de certaines de ses parties et entraîner des inconsistances sur les annotations sémantiques reposant sur cette ontologie de référence.

Notre approche consiste à construire un système de gestion de l'évolution d'un Web sémantique d'entreprise CoSWEM dont le but est de surveiller et de gérer l'évolution de l'ontologie et de l'annotation sémantique. Du côté de l'ontologie, ce système permet à l'utilisateur de comparer les différences entre deux versions de l'ontologie. Grâce aux interfaces graphiques du CoSWEM, nous pouvons visualiser les changements effectués entre deux versions de l'ontologie ainsi que les éléments affectés par les changements ontologiques. De plus, nous pouvons aussi déterminer quelles sont les annotations sémantiques affectées par les changements capturés dans une trace d'évolution de l'ontologie. Du point de vue de l'annotation sémantique, le système CoSWEM aide à la résolution de deux tâches importantes : (i) la détection des inconsistances générées sur les annotations sémantiques à cause des modifications de l'ontologie de base et (ii) la correction de ces annotations sémantiques inconsistantes en vue d'assurer la consistance globale du système de gestion des connaissances. Ces tâches sont réalisées d'une

manière automatique ou semi-automatique avec l'aide de l'utilisateur (i.e. l'ontologiste, l'annotateur…)

Récapitulons les principales étapes du processus d'élaboration de notre approche et du système CoSWEM. Le noyau du système CoSWEM est le composant d'évolution qui se compose de trois sous-composants (i) l'évolution d'ontologie, (ii) l'évolution d'annotation sémantique et (iii) l'évolution de ressource. Dans le cadre de cette thèse, nous sommes focalisés sur la réalisation des points (i) et (ii) qui sont dédiés à la gestion de l'évolution de deux des composants les plus évolués d'un Web sémantique d'entreprise, i.e. l'ontologie et l'annotation sémantique respectivement. D'autre part, CoSWEM a quelques fonctions qui ont pour but de communiquer avec l'utilisateur (i.e. notifier des messages, des résultats…) et de gérer les parties du système de gestion des connaissances (i.e. charger l'ontologie et la base d'annotations, charger la trace d'évolution…). Au cours de l'évolution, des changements doivent être identifiés et représentés dans les formats appropriés. Nous avons classé les différents types de changements de l'ontologie selon des critères différents tels que le niveau de granularité (i.e. changement élémentaire ou composite), l'affectation sur l'annotation (i.e. correction obligatoire ou facultative) ou l'élément affecté (i.e. changement sur le concept ou sur la propriété). Après avoir appliqué des changements sur une ontologie, cette ontologie a évolué vers une nouvelle version. Le système CoSWEM aide à deux cas d'évolution de l'ontologie qui peuvent influencer l'état consistant de l'annotation : (a) avec trace et (b) sans trace des changements effectués (ou trace d'évolution) entre deux versions de l'ontologie.

Correspondant aux deux scénarios d'évolution de l'ontologie mentionnés ci-dessus, nous proposons deux approches (i) une approche procédurale et (ii) une approche basée sur des règles, qui sont respectivement dédiées à la réalisation de l'évolution de l'annotation dans les deux cas d'évolution.

Un autre avantage de notre approche est l'intégration du moteur de recherche sémantique Corese dans ce travail de thèse. Corese nous permet de lancer des requêtes selon différentes exigences sur la hiérarchie des concepts et des propriétés de l'ontologie ainsi que sur la base d'annotations. Il se différencie des moteurs de recherche syntaxiques basés sur la technique de mots-clés. Corese effectue quant à lui une recherche sémantique. Parmi les tâches spécifiques du CoSWEM, le moteur Corese facilite la recherche des éléments qui diffèrent ou ceux qui sont affectés et obsolètes par rapport aux changements ontologiques effectués.

D'autre part, nous avons implémenté dans CoSWEM des algorithmes pour réaliser quelques tâches spécifiques. Par exemple les algorithmes, combinés avec le moteur de recherche Corese, pour détecter des éléments inconsistants sur le concept, la propriété, le domaine… Pour le processus de résolution semi-automatique de l'inconsistance sur les annotations, nous avons implémenté l'algorithme permettant de choisir un "meilleur" élément de la nouvelle version de

l'ontologie pour remplacer celui qui est obsolète et inconsistant dans l'annotation sémantique.

Enfin, nous faisons l'expérimentation et l'évaluation du système CoSWEM dans des scénarios réels. Cette expérimentation a été appliquée dans le cadre des projets PALETTE et E-WOK_HUB en validant les fonctions principales de ce système.

Revue des questions de recherche

Nous rappelons dans cette section la problématique et particulièrement les questions de recherche que nous avons listées au début de ce manuscrit. Nous réaffirmons que les caractéristiques principales telles que l'environnement dynamique, collaboratif, hétérogène et distribué de la prochaine génération du Web sémantique… renforceront clairement le besoin de résoudre la problématique de changement et de l'évolution. Cette évolution devra être bien gérée et contrôlée dans la nouvelle infrastructure du Web sémantique.

Ce manuscrit a apporté certaines contributions aux recherches sur la gestion de l'évolution du système de gestion des connaissances. Elles peuvent être manifestées à travers des réponses aux questions de recherche suivantes :

Question 1 : Quels sont les types de changements possibles et leurs affectations au système de gestion des connaissances ?

Le travail de thèse s'est concentré sur l'évolution des deux composants principaux du Web sémantique d'entreprise (i.e. ontologie et annotation sémantique) et au contexte d'évolution : l'ontologie est souvent modifiée en vue de s'adapter aux nouveaux besoins de l'environnement et les changements de l'ontologie peuvent affecter la consistance des annotations sémantiques reposants sur cette ontologie. Nous nous focalisons donc d'abord sur les types de changements de l'ontologie.

Nous avons construit une classification de 26 changements ontologiques selon les critères suivants (i) le niveau d'abstraction (les changements élémentaires ou composites), (ii) l'affectation sur l'annotation (les corrections obligatoires ou facultatives) et (iii) l'élément affecté (les changements sur le concept ou sur la propriété). D'autre part, nous avons élaboré une ontologie d'évolution qui a pour but de modéliser d'une manière formelle ces types de changements et les informations concernant le processus d'évolution. Cette ontologie d'évolution sert également à représenter la trace d'évolution qui capture tous les changements effectués entre deux versions de l'ontologie.

Question 2 : Comment peut-on détecter les inconsistances générées à cause des changements de l'ontologie?

Nous avons montré des exemples réels qui justifient que la modification de l'ontologie peut entrainer des inconsistances au niveau des annotations sémantiques par rapport à la nouvelle version de l'ontologie. Puisqu'il est irréaliste pour l'ontologiste de bien connaître toutes les parties de l'ontologie et ainsi savoir celles qui sont affectées (i.e. les éléments relatifs, les annotations concernées, etc.) par les changements de cette ontologie, la détection d'inconsistance devrait alors être réalisée automatiquement.

Au cours de l'évolution de l'ontologie, nous trouvons qu'il existe deux contextes principaux qui peuvent être distingués par le fait que la trace d'évolution soit conservée ou pas. Dans le premier cas où l'on peut garder la trace d'évolution entre les deux versions O_1 et O_2 de l'ontologie, on conserve tous les changements effectués ainsi que les résultats des opérations. Cette trace de changement est formalisée à l'aide d'une annotation sémantique reposant sur une ontologie d'évolution qui décrit d'une manière formelle les types de changements ontologiques possibles ainsi que les informations concernant l'exécution de changement. Le deuxième cas quant à lui est fréquent dans le contexte dynamique et distribué du Web sémantique. En effet, dans ce contexte, il n'est pas toujours possible de garder la trace d'évolution entre des versions de l'ontologie. CoSWEM supporte donc dans ce cas, une fonction permettant de comparer les différences entre deux versions de l'ontologie au niveau les entités de concept et de propriété.

Pour la gestion de l'évolution de l'annotation sémantique, l'approche procédurale est appliquée si les changements de l'ontologie effectués sont connus (i.e. avec trace). Nous utilisons le moteur de recherche Corese pour exploiter les informations relatives sauvegardées dans la trace d'évolution en vue de trouver des annotations vraies inconsistantes. Nous appliquons ensuite des stratégies de résolution pour corriger ces inconsistances détectées. Pour chacun des changements ontologiques qui pourrait affecter la consistance des annotations, nous avons construit une stratégie équivalente afin de corriger les inconsistances apparues dans les annotations sémantiques suite à ce type de changement. Au contraire, si la trace d'évolution n'est pas gardée, l'approche basée sur des règles est lancée. Nous appliquons des règles de détection d'inconsistance pour chercher des annotations qui deviennent obsolètes et inconsistantes par rapport à la nouvelle version de l'ontologie. Ces règles sont élaborées à partir des contraintes de consistance qui doivent être satisfaites pour le modèle d'annotation. Dans ce cas, nous pouvons réaliser la tâche de résolution des annotations inconsistantes d'une manière semi-automatique, c.-à-d. avec l'aide de l'utilisateur auquel on propose de choisir une opération qui lui convienne parfaitement (i.e. supprimer, remplacer, laisser sans modification…).

Question 3 : Comment résoudre les inconsistances détectées à cause de la modification de l'ontologie pour assurer la consistance globale du système?

Pour la résolution des annotations inconsistantes détectées, nous avons proposé deux méthodes : méthode automatique ou semi-automatique. Dans le cas

où on connaît comment les changements de l'ontologie ont été résolus et les stratégies de résolution appliquées pour l'ontologie, nous pouvons utiliser des règles de correction afin de réaliser automatiquement la stratégie de résolution choisie pour l'annotation sémantique qui a été construite respectivement pour chaque stratégie de l'ontologie. D'autre part, si nous n'avons aucune information sur l'évolution de l'ontologie, nous pouvons réaliser la tâche de résolution des annotations inconsistantes d'une manière semi-automatique avec l'aide de l'utilisateur pour choisir une opération qui lui convient.

Question 4 : Comment appliquer et exploiter ces recherches sur l'évolution à un problème réel qui demande la gestion de l'évolution? Est-il possible de développer des outils de support pour gérer l'évolution du système de gestion des connaissances?

Toutes les propositions de la nouvelle approche proposée dans cette thèse ont été implémentées et validées dans le système de gestion de l'évolution du Web sémantique d'entreprise intitulé CoSWEM. Ce système facilite, pour l'utilisateur, la gestion de l'évolution en lui proposant de réaliser certaines tâches d'une manière automatique ou semi-automatique telles que la comparaison des ontologies différentes, la détection et la correction des inconsistances sur les annotations sémantiques, etc. Dans un premier temps, le système CoSWEM est également expérimenté dans le cadre des projets PALETTE et E-WOK_HUB avec un ensemble de données réelles et évolutives provenant de ces projets ainsi que dans des scénarios d'évolution qui sont fréquents dans la réalité : (i) avec trace d'évolution et (ii) sans trace d'évolution de l'ontologie.

Limites et difficultés

Le travail présenté dans ce manuscrit ne prétend pas être une solution complète répondant à tous les aspects différents du problème d'évolution du Web sémantique d'entreprise. Il reste encore quelques limites et difficultés dans la construction du système de gestion de l'évolution CoSWEM :

- *Types de changements de l'ontologie :* Bien que nous ayons construit un ensemble de types de changements différents qui peuvent couvrir la plupart des cas de changements usuels, certains changements complexes ne sont pas encore résolus d'une manière automatique. Par exemple pour un concept c qui a plusieurs super-concepts différents, si on supprime ce concept c, on ne sait pas comment rebrancher automatiquement les sous-concepts de c à un super-concept (il n'est pas réaliste de rebrancher d'une manière automatique tous les sous-concepts de c à chaque super-concept de c). On a le même problème dans le cas où une propriété *p* possédant plusieurs concepts domaines est enlevée. Si on veut diviser cette propriété *p* en deux nouvelles propriétés (*p* est supprimée après la division), on ne peut pas

déterminer automatiquement à quel concept (parmi les concepts domaines de *p*) rattacher chacune des nouvelles propriétés. A notre connaissance, aucun outil ni méthode existante ne traite cette difficulté et il serait irréaliste de décider de manière automatique un choix de stratégies de résolution correspondant à ces types de changements complexes. C'est une difficulté de notre approche dans la résolution d'inconsistances.

- *Format de représentation de l'annotation sémantique :* Dans le but d'illustrer l'évolution de l'annotation sémantique, nous avons choisi et implémenté un format d'annotation sémantique en RDF dans le système CoSWEM selon les normes proposées par le W3C. Cependant, cette norme permet d'écrire une même relation entre instances de différentes manières. Or, s'il existe plusieurs formats différents de l'annotation sémantique dans la base d'annotations, par exemple une même ressource d'annotation peut être représentée en deux formes différentes `<rdfs:Class rdf :ID="Personne">` ou `<rdf :Description rdf:ID="Personne"><rdf:type rdf:ressource="http://www.w3.org/2000/01/rdf-schema#Class"></rdf :Description>`. Cela empêchera la détection et la résolution des annotations inconsistantes.

- *Traitement distribué :* Dans un premier temps, nous travaillons principalement avec une mémoire d'entreprise locale et centralisée. Nous nous sommes donc focalisés sur le développement d'un système de gestion de l'évolution du Web sémantique d'entreprise dans un contexte centralisé. Le système CoSWEM ne supporte pas actuellement le traitement de l'évolution de l'ontologie et des annotations sémantiques dans un environnement distribué.

Perspectives

Dans cette section, nous présentons les possibilités d'étendre et d'approfondir les recherches réalisées dans ce manuscrit en vue d'un travail futur. Ce travail peut être divisé en perspectives à court terme et à long terme.

- ***Perspectives à court terme***

  - Il serait intéressant d'élaborer un mécanisme permettant à l'utilisateur de choisir de manière semi-automatique des solutions pour résoudre les contextes de changements complexes mentionnés ci-dessus : par exemple le système pourrait calculer les meilleures stratégies à appliquer, puis l'utilisateur choisirait une stratégie de résolution selon sa préférence.

- Nous irons automatiser la tâche de résolution des annotations sémantiques inconsistantes dans le cas où le système connaît la stratégie de résolution correspondante appliquée pour l'ontologie.

- Enfin, il faudrait intégrer des stratégies de résolution pour tous les types de changements ontologiques dans l'éditeur d'ontologie ECCO développé dans le cadre du projet E-WOK_HUB.

- ***Perspectives à long terme***

- Comme le montrait dans le chapitre 2, l'évolution des ressources peut aussi entraîner des inconsistances sur la base d'annotations décrivant ces ressources. Il serait intéressant d'étudier les types de changements des sources de documents, des informations contextuelles et leur influence sur la base d'annotations existante ainsi que le processus d'annotation à partir de ces sources.

- En se basant sur les fonctionnalités principales du CoSWEM et des autres outils développés dans l'équipe, nous visons à construire un système global de gestion du Web sémantique d'entreprise qui combine plusieurs modules différents pour servir à la gestion des connaissances d'un entreprise : outil de création et de modification de l'ontologie et de l'annotation sémantique, outils de réutilisation des ontologies et des bases d'annotations existantes, outil de détection et de résolution des inconsistances, etc.

- Nous avons choisi RDF(S) pour développer l'ontologie et les annotations sémantiques dans notre travail. Cependant, nous pensons également au formalisme OWL qui est utilisé largement pour représenter des ontologies. La grande d'expressivité du langage OWL impliquera aussi d'autres types de changements de l'ontologie possibles (i.e. changement de cardinalité, changement d'une restriction sur un concept…). Il est nécessaire de proposer des solutions pour résoudre les inconsistances d'annotations générées par les nouvaux changements.

- Il existe actuellement plusieurs travaux sur la génération des annotations sémantiques ainsi que l'enrichissement de l'ontologie du domaine avec les résultats du processus d'annotation. Dans ce cas, l'évolution de l'annotation sémantique influencera également les ontologies. Ce serait une perspective intéressante pour la recherche sur l'évolution.

- Dans le futur du Web sémantique, un système de gestion de connaissance sera élaboré et exploité par plusieurs personnes ou par plusieurs organisations d'une manière collaborative. Le système CoSWEM doit donc être étendu pour fonctionner dans un tel environnement distribué.

Bibliographie

[Adida et Birbeck, 2007] Adida, B. et Birbeck, M. (2007). RDFa Primer 1.0 – Embedding RDF in XHTML. W3C Working Draft 12 March 2007. Download available at http://www.w3.org/TR/xhtml-rdfa-primer/

[Amardeilh, 2007] Amardeilh, F. (2007). Web Sémantique et Informatique Linguistique : propositions méthodologiques et réalisation d'une plateforme logicielle. Thèse de doctorat. Discipline : Informatique . Université Paris X – Nanterre, Mai 2007

[Aussenac-Gilles et al., 2000] Aussenac-Gilles N., Biebow B. et Szulman S, (2000). Revisiting ontology design: a method based on corpus analysis, in Proceedings of the conference EKAW'2000, Springer LNCS 1937, pages 172-188, 2000.

[Banerjee et al., 1987] Banerjee, J., Kim, W., Kim, H.J. et Korth, H. (1987). Semantics and implementation of schema evolution in object-oriented databases. In Proceedings of the Annual Conference on Management of Data (ACM SIGMOD 16(3)), San Francisco, pp. 311-322, 1987.

[Bechhofer et al., 2001] Bechhofer, S., Horrocks, I., Goble, C. et Stevens, R. (2001). OilEd: A reasonable ontology editor for the semantic web. In KI-2001: Advances in Artificial Intelligence, LNAI 2174, pages 396–408. Springer, 2001.

[Benayache, 2005] Benayache, A. (2005). Construction d'une memoire organisationnelle de formation et evaluation dans un contexte e-learning: le projet Memorae. Université de Technologie de Compiègne. Spécialité: Technologies de l'Information et des Systèmes (TIS).

[Berners-Lee et al., 2001] Berners-Lee, T., Hendler, J. et Lassila, O. (2001). The Semantic Web, In Scientific American, May 2001, p35-43

[Bhide et al., 2002] M. Bhide, P. Deoasee, A. Katkar, A. Panchbudhe and K. Ramamritham. Adaptive push-pull: disseminating dynamic Web data. IEEE Transaction on Computers, Volume 51, Number 6, pp. 652-668, June 2002.

[Boley et Kifer, 2007] Boley, H. et Kifer, M. (2007). RIF Core Design - W3C Working Draft 30 March 2007. Download available at http://www.w3.org/TR/2007/WD-rif-core-20070330.

[Bounaas, 1995] Bounaas , F. (1995). Gestion de l'évolution dans les bases de connaissances : une approche par les règles. Mémoire de thèse. L'Institut National Polytechnique de Grenoble.

[Breche et Woerner, 1995] Breche, P., Woerner. M. (1995). How to remove a class in an object database system. In Proceedings of the 2nd International Conference on Applications of Databases (ADBS.95), San Jose, California, pp. 235-246, 1995.

[Brickley et Guha, 2004] Brickley, D. et Guha, R.V. (2004). RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation 10 February 2004. Available download at http://www.w3.org/TR/2004/REC-rdf-schema-20040210/

[Broekstra et al., 2002] Broekstra, J., Kampman, A. et van Harmelen, F. (2002). Sesame: A Generic Architecture for Storing and Querying rdf and rdf Schema. In The Semantic Web - ISWC 2002.

[Bruijn et al., 2004] Bruijn, J., Martin-Recuerda, F., Manov, D. et Ehrig, M. (2004). State-of-the-art survey on Ontology Merging and Aligning V1. Deliverable D4.2.1 (WP4). EU-IST Integrated Project (IP) IST-2003-506826 SEKT.

[Ceravolo et al., 2004] Ceravolo, P., Corallo, A., Elia, G. et Zilli, A. (2004). Managing Ontology Evolution Via Relational Constraints. Knowledge-Based Intelligent

Information and Engineering Sys-tems, 8th International Conference, KES 2004, Wellington, New Zealand. Lecture Notes in Computer Science 3215, Springer 2004

[Charlet et al., 2003] Charlet, J., Laublet, P. et Reynaud, C. (2003). Web sémantique - Rapport final V3 – décembre 2003. Action spécifique 32 CNRS / STIC (http://www.lalic.paris4.sorbonne.fr/stic/).

[Charlet, 2000] Charlet, J. (2003). L'Ingénierie des Connaissances : Développements, Résultats et Perspectives pour la gestion des connaissances médicales. Mémoire d'Habilitation à diriger des recherches – Université Pierre et Marie Curie, 2003.

[Claypool et Rundensteiner, 1998] Claypool, K. et Rundensteiner, E. (1998). SERF: Schema Evolution through an Extensible, Re-usable and Flexible Framework. International Conference on Information and Knowledge Management, Bethesda, Maryland, USA, November 1998.

[Cobéna, 2003] Cobéna, G. (2003). Change management of semi-structured data on the Web. Thèse d'Informatique, spécialité Algorithmique. Ecole Polytechnique.

[Corby et al, 2006] Corby, O., Dieng-Kuntz, R., Faron-Zucker, C. et Gandon, F. (2006). Searching the Semantic Web: Approximate Query Processing based on Ontologies IEEE Intelligent Systems Journal, January/February 2006 (Vol. 21, No. 1).

[Corby et al., 2004] 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, Proc. of the 16th European Conference on Artificial Intelligence (ECAI'2004), Valencia, 22-27 August 2004, IOS Press, p.705-709.

[Corby et Faron, 2007] Corby, O. et Faron, C. (2007). Implementation of SPARQL Query Language based on Graph Homomorphism. In Proc. of the 15th International Conference on Conceptual Structures (ICCS'2007), Sheffield, UK.

[Corcho et al., 2002] Corcho, O., Fernández-López, M., Gómez-Pérez, A. et Vicente, O. (2002). WebODE: an integrated workbench for ontology representation, reasoning and exchange. 13th International Conference on Knowledge Engineering and Knowledge Management EKAW02.

[Corcho et Gómez-Pérez, 2000] Corcho, O. et Gómez-Pérez, A. (2000). A Roadmap to Ontology Specification Languages. In R. Dieng et O. Corby (éd.), Knowlege Engineering and Knowledge Management Methods, Models, and Tools. Poceedings of EKAW 2000, 2000, pp.80-96.

[Davies et al., 2002] Davies, J., Fensel, D. et van Harmelen, F. (2002). Towards the Semantic Web: Ontology-driven Knowledge Management. John Wiley & Sons, Ltd, England 2002. ISBN: 978-0-470-84867-8.

[Dieng-Kuntz et al., 2004] Dieng-Kuntz, R., Minier, D., Corby, F., Ruzicka, M., Corby, O., Alamarguy, L. et Luong, P-H (2004). Medical Ontology and Virtual Staff for a Health Network, In Enrico Motta and Nigel Shadboldt eds, Engineering Knowledge in the Age of the Semantic Web: Proceedings of the 14th International Conference, EKAW 2004, Whittlebury Hall, UK, October 5-8, 2004, LCNS: Springer-Verlag Heidelberg, Volume 3257, ISBN: 3-540-23340-7, pp. 187 - 202.

[Dieng-Kuntz et al., 2005] Dieng-Kuntz, R., Corby, O., Gandon, F., Giboin, A., Golebiowska, J., Matta, N. et Ribière, M. (2005). Knowledge Management-Methodes et outils pour la gestion des connaissances; une approche pluridisciplinaire du Knowledge Management. 2e dition, Dunod, Paris, 2005.

[Dieng-Kuntz, 2005] Dieng-Kuntz, R. (2005). Corporate Semantic Webs. In Encyclopaedia of Knowledge Management, D. Schwartz ed, Idea Publishing Group, September 2005.

[Ermine, 2000] Ermine J.-L. (2000). Enjeux, démarches et processus de la gestion des connaissances. In Actes des journées francophones d' Ingénierie des Connaissances (IC' 2000) - Conférence tutorielle, Toulouse, France.

[Fensel et al., 2002] Fensel, D., van Harmelen, F. et Horrocks, I. (2002). OIL & DAML+OIL: Ontology languages for the Semantic Web. In J. Davis et al. (eds.), Towards the Semantic Web: Ontology-Driven Knowledge Mangement, Wiley, 2002.

[Fernandez et al., 1997] Fernandez M., Gomez-Perez A. et Juristo N. (1997). METHONTOLOGY : from ontological art towards ontological engineering, in Proceedings of the Spring Symposium Series on Ontological Engineering (AAAI'97), AAAI Press.

[Flouris, 2006] Flouris, G. (2006). On Belief Change and Ontology Evolution. PhD thesis, University of Crete, Department of Computer Science.

[Franconi et al., 2000] Franconi, E., Grandi, F. et Mandreoli, F. (2000). A semantic approach for schema evolution and versioning in object-oriented databases. Computational Logic 2000, pp. 1048-1062, 2000.

[Gabel et al., 2004] Gabel, T., Sure, Y. et Voelker, J. (2004). KAON – Ontology Management Infrastructure. SEKT deliverable D3.1.1.a. http://km.aifb.unikarlsruhe.de/projects/sekt/index_html

[Gandon et Durville, 2007] Gandon, F. et Durville, P. (2007). SeWeSe: Semantic Web Server. The 16th International World Wide Web Conference (WWW2007). May 8-12, 2007. Banff, Alberta, Canada.

[Gandon, 2002] F.Gandon. Distributed Artificial Intelligence and Knowledge Management: Ontologies and Multi-Agent System for a Corporate Semantic Web. PhD Thesis at INRIA Sophia Antipolis, ACACIA team.

[Gennari et al., 2003] Gennari, J.H., Musen, M.A., Fergerson, R.W., Grosso, W.E., Crubezy, M., Eriksson, H., Noy, N.F., et Tu, S.W. (2003) The evolution of Protégé: an environment for knowledge-based systems development. In: International Journal of Human-Computer Studies, 2003, 58 (1) pp.89-123.

[Gil et Tallis, 1997] Gil, Y. et Tallis, M. (1997). A Script-Based Approach to Modifying Knowledge Bases. In Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97), Providence, RI, pp. 377-383, 1997.

[Golebiowska et al., 2001] Golebiowska, J., Dieng-Kuntz, R., Corby, O. et Mousseau, D. (2001). Building and Exploiting Ontologies for an Automobile Project Memory. First International Conference on Knowledge Capture (K-CAP), Victoria, October 23–24.

[Gomez-Perez et al., 2002] Gomez-Perez, A., Angele, J., Fernandez-Lopez, M., Christophides, V., Stutt, A. et Sure, Y. (2002). A survey on ontology tools. OntoWeb - Ontology-based information exchange for knowledge management and electronic commerce, IST-2000-29243. Deliverable 1.3, Universidad Politecnica de Madrid.

[Gómez-Pérez et al., 2004] Gómez-Pérez, A., Fernández-López, M. et Corcho, O. (2004). Ontological Engineering, with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web. Springer Verlag London 2004. ISBN 1852335513.

[Gruber, 1993] Gruber, R. (1993). A translation approach to portable ontology specifications, Knowledge Acquisition, An International Journal of Knowledge Acquisition for Knowledge-Based Systems, Volume 5, Number 2, pp.199-220.

[Gruninger et Fox, 1995] Gruninger M. et Fox M. S., (1995). Methodology for the design and evaluation of ontologies, in Proceedings of the Workshop on Basic Ontological Issues on Knowledge Sharing, IJCAI'95.

[Haase et al., 2005] Haase, P., van Harmelen, F., Huang, Z., Stuckenschmidt, H. et Sure, Y. (2005). A framework for handling inconsistency in changing ontologies. In Y. Gil, E. Motta, V. Benjamins, and M. Musen, editors, Proceedings of the 4th International Semantic Web Conference (ISWC '05), volume 3729 of Lecture Notes in Computer Science, pages 353–367. Springer.

[Haase et Stojanovic, 2005] Haase, P. et Stojanovic, L. (2005). Consistent evolution of OWL ontologies. In A. Gomez-Perez and J. Euzenat, editors, Proceedings of the 2nd

European Semantic Web Conference (ESWC ’05), volume 3532 of LCNS, pages 182–197. Springer, 2005.

[Haase et Sure, 2004] Haase, P. et Sure, Y. (2004). D3.1.1.b State-of-the-Art on Ontology Evolution. SEKT/2004/D.3.1.1.b/v0.5. Institute AIFB, University of Karlsruhe.

[Handschuh et al., 2001]Handschuh, S., Staab, S. et Maedche, A. (2001). CREAM - Creating Relational Metadata with a Component-Based, Ontology-Driven Framework. K-CAP 2001, First International Conference on Knowledge Capture, Oct. 21-3, 2001, Victoria, B.C., Canada.

[Handschuh et al., 2002] Handschuh S., Staab S. et Ciravegna F. (2002). S-CREAM – Semiautomatic CREAtion of Metadata.The 13th International Conference on Knowledge Engineering and Management (EKAW 2002), ed Gomez-Perez, A., Springer Verlag.

[Handschuh et al., 2003] Handschuh S., Staab S. et Studer R. (2003). Leveraging metadata creation for the Semantic Web with CREAM, KI '2003 - Advances in Artificial Intelligence. In Proceedings Annual German Conference on AI, Sept. 2003.

[Handschuh, 2005] – Handschuh S. (2005). Creating Ontology-based Metadata by Annotation for the Semantic Web. Thèse de doctorat, Université de Karlsruhe, 2005, 225 p.

[Heflin et Hendler, 2000] Heflin, J. et Hendler, JA. (2000). Dynamic Ontologies on theWeb. Proceedings of the 7th National Conference on Artificial Intelligence AAAI-2000, Menlo Park, CA, August 2000, pp 443-449. AAAI/MIT Press, Cambridge, MA.

[Heflin, 2001] J.Heflin. Towards the Semantic Web: Knowledge Representation in a Dynamic, Distributed Environment. PhD Thesis. Faculty of the Graduate School of the University of Maryland.

[Horrocks et al., 2000] Horrocks, I., Fensel, D., Broekstra, J., Decker, S., Erdmann, M., Goble, C., van Harmelen, F., Klein, M., Staab, S., Studer, R., et Motta, E. (2000). OIL: The Ontology Inference Layer. Technical Report IR-479, Vrije Universiteit Amsterdam, Faculty of Sciences. See http://www.ontoknowledge.org/oil/.

[Huang et Stuckenschmidt, 2005] Huang, Z. et Stuckenschmidt, H. (2005). Reasoning with multi-version ontologies: a temporal logic approach. In Y. Gil, E. Motta, V. Benjamins, and M. Musen, editors, Proceedings of the 4th International Semantic Web Conference (ISWC ’05), volume 3729 of Lecture Notes in Computer Science, pages 398–412. Springer, 2005.

[IEEE, 1990] IEEE 1990. Institute of Electrical and Electronics Engineers, IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. New York, NY, 1990.

[Kahan et al., 2001] Kahan J., Koivunen M-J., Prud'Hommeaux E. et Swick R. (2001). Annotea: An Open RDF Infrastructure for Shared Web Annotations. In Proceedings 10th International World Wide Web Conference (WWW 2001) Hong Kong.

[Kassel, 2002] Kassel, G. (2002). OntoSpec : une méthode de spécification semi-informelle d’ontologies. Actes de IC’2002, Rouen, Mai 2002.

[Kiryakov et Ognyanov, 2002] Kiryakov, A. et Ognyanov, D. (2002). Tracking changes in RDF(S) repositories, in Proceedings of 13 International Conference on Knowledge Engineering and Knowledge Management (EKAW 2002), Siguenza, Spain, LNCS 2473, pp. 373-378, 2002.

[Klein et al., 2002] Klein, M., Kiryakov, A., Ognyanov, D. et Fensel, D. (2002). Ontology versioning and change detection on the Web. Proceedings of the 13th European Conference on Knowledge Engineering and Knowledge Management (EKAW-2002), Siguenza, Spain, October 2002, pp 197-212.

[Klein et al., 2002a] Klein, M., Fensel, D., Kiryakov, A. et Ognyanov, D. (2002). Ontoview: Comparing and Versioning Ontologies. In Collected Posters ISWC 2002, Sardinia, Italy.

[Klein et Fensel, 2001] Klein, M. et Fensel, D. (2001). Ontology versioning for the Semantic Web. In Proceedings of the First International Semantic Web Working Symposium (SWWS), pages 75-91, Stanford University, California, USA.

[Klein et Noy, 2003] Klein, M. et Noy, N.F. (2003). A component-based framework for ontology evolution. In Proceedings of the Workshop on Ontologies and Distributed Systemsm, IJCAI '03, Acapulco, Mexico. Also available as Technical Report IR-504, Vrije Universiteit Amsterdam.

[Klein, 2004] Klein, M. (2004). Change Management for Distributed Ontologies. PhD thesis, Vrije Universiteit Amsterdam, Aug. 2004.

[Klyne et Carroll, 2004] Klyne, G. et J. Carroll (2004). Resource description framework (rdf) : Concepts and abstract syntax. Technical report, In W3C Recommendation. (http ://www.w3.org/RDF).

[Koivunen, 2005] Koivunen M-R., (2005). Annotea and Semantic Web Supported Collaboration, Invited talk at Workshop on User Aspects of the Semantic Web (UserSWeb), at European Semantic Web Conference (ESWC 2005) Heraklion, Greece, 29 May 2005.

[Kuhn et Abecker, 1997] Kuhn, O. et Abecker, A. (1997) Corporate memories for Knowledge Management in Industrial Practice: Prospects and Challenges, Journal of Universal Computer Science, vol. 3(8): 929-954, 1997.

[Laublet et al., 2002] Laublet, P., Reynaud, C., et Charlet, J. (2002) Sur quelques aspects du web sémantique. Actes des deuxièmes assises du GdR 13, pp 59-78, Cépaduès Editions, 2002.

[Liang et al., 2006] Liang, Y., Alani, H., Dupplaw, D. et Shadbolt, N. (2006). An Approach to Cope with Ontology Changes for Ontology-based Applications. In Proceedings of Second Advanced Knowledge Technologies DTA Symposium, Aberdeen.

[Lindgren et al., 2000] Lindgren, R., Hardless, C., Nulden, U. et Pessi, K. (2000). The Evolution of knowledge management system need to be managed, http://www.viktoria.informatik.gu.se/groups/ KnowledgeManagement/Documents/kmman.pdf, 2000.

[Luong et al., 2006] Luong, P-H., Dieng-Kuntz, R. et Boucher, A. (2006). Managing semantic annotations evolution in the CoSWEM system. Proceedings of the Third National Symposium on Research, Development and Application of Information and Communication Technology (ICT.rda'06), Hanoi (Vietnam), May 2006, pages 215 – 223.

[Luong et al., 2007] Luong, P-H., Dieng-Kuntz, R. et Boucher, A. (2007). Evolution de l'ontologie et gestion des annotations sémantiques inconsistantes. 7ème conférence francophone Extraction et Gestion des Connaissances (EGC'2007), Janvier 2007, Belgique, pp.635-646.

[Luong et Dieng-Kuntz, 2006] Luong, P-H. et Dieng-Kuntz, R. (2006). A Rule-based Approach for Semantic Annotation Evolution in the CoSWEM System. In Proceeding of the Canadian Semantic Web Working Symposium (CSWWS 2006), Quebec city, Canada, June 2006, pages 103-120.

[Luong et Dieng-Kuntz, 2007] Luong, P-H. et 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, August 2007.

[Luong et Dieng-Kuntz, 2005] Luong, P-H. et Dieng-Kuntz, R. (2005). Evolution Management System for a Corporate Semantic Web, Conference YVSM 2005 (Young Vietnamese Scientists Meeting 2005), June 12-16, 2005 in Nha Trang, Vietnam.

[Maedche et al., 2003] Maedche, A., Motik, B. et Stojanovic, L. (2003). Managing multiple and distributed ontologies on the semantic web. The VLDB Journal, 12:286-302.

[Maedche et Staab, 2003] Maedche, A. et Staab, S. (2003). KAON: The Karlsruhe Ontology and Semantic Web Meta Project.. Künstliche Intelligenz (3): 27-30. July 2003. Special issue on Semantic Web

[McBride, 2001] McBride, B. (2001). Jena: Implementing the RDF Model and Syntax Specification. Semantic Web Workshop, WWW2001. http://www.hpl.hp.com/personal/bwm/papers/20001221-paper/

[Menzies, 1999] T. Menzies. Knowledge maintenance: The state of the art. The Knowledge Engineering Review, Volume 14, Number 1, pp. 1-46, 1999.

[Miller et Manola, 2004] Miller, E. et Manola, F. (2004). RDF Primer. W3C Resource Description Framework, available at http://www.w3.org/RDF.

[Mougin, 2006] Mougin, F. (2006). Conception d'un modèle Web sémantique appliqué à la génomique fonctionnelle. Mémoire de thèse. Université de Rennes I.

[Noy et al., 2000] Noy, N., Fergerson, R. et Musen, M. (2000). The knowledge model of Protégé-2000: Combining interoperability and flexibility. In R. Dieng and O. Corby, editors, Proceedings of the 12th International Conference on Knowledge Engineering and Knowledge Management: Methods, Models, and Tools (EKAW 2000), volume 1937 of Lecture Notes in Artificial Intelligence (LNAI), pages 17–32, Juan-les-Pins, France, 2000. Springer.

[Noy et al., 2004] Noy, N.F., Kunnatur, S., Klein, M. et Musen, M.A. (2004). Tracking Changes During Ontology Evolution. In Proceedings of the 3rd International Conference on the Semantic Web (ISWC-04), published as Lecture Notes in Computer Science (LNCS), Volume 3298, pages 259-273, S.A. McIlraith, D. Plexousakis, F. van Harmelen (eds), Springer-Verlag, 2004.

[Noy et Klein, 2004] Noy, N.F. et Klein, M. (2004). Ontology evolution: Not the same as schema evolution. Knowledge and Information Systems, 6(4):428-440.

[Noy et Musen, 2002] Noy, N.F. et Musen, M.A. (2002). PROMPTDIFF: A Fixed-Point Algorithm for Comparing Ontology Versions. Proc. 18th Nat'l Conf. Artificial Intelligence (AAAI 2002), AAAI Press, 2002, pp. 744–750.

[Noy et Musen, 2003a] Noy, N. F., et Musen, M. A. (2003). The PROMPT Suite: Interactive Tools For Ontology Merging And Mapping International Journal of Human-Computer Studies, 59/6 pp. 983-1024. Available as SMI technical report SMI-2003-0973

[Noy et Musen, 2004] Noy, N.F., Musen, M.A. (2004). Ontology Versioning in an Ontology Management Framework. IEEE Intelligent Systems, Vol. 19, No. 4, July – August, 2004.

[Peters et Oezsu, 1997] Peters, R. J. et Oezsu, M. (1997). An axiomatic model of dynamic schema evolution in object-base management systems. ACM Transactions on Database Systems, Volume 22, Number 1, pp. 75-114, 1997.

[Pinto et al., 2004] Pinto, H.S., Tempich, C. et Staab, S. (2004). DILIGENT: Towards a fine-grained methodology for DIstributed, Loosely-controlled and evolvInG Engingeering of oNTologies. In Ramon Lopez de Mantaras and Lorenza Saitta, Proceedings of the 16th European Conference on Artificial Intelligence (ECAI 2004), August 22nd - 27th, pp. 393--397. IOS Press, Valencia, Spain, August 2004. ISBN: 1-58603-452-9. ISSN: 0922-6389

[Plessers et Troyer, 2005] Plessers, P. et De Troyer, O. (2005). Ontology Change Detection using a Version Log. Proceedings of the 4th International Semantic Web Conference, pp. 578-592, Eds. Yolanda Gil, Enrico Motta, V.Richard Benjamins, Mark A. Musen, Publ. Springer-Verlag, ISBN 978-3-540-29754-3, Galway, Ireland.

[Plessers, 2006] Plessers, P. (2006). An Approach to Web-based Ontology Evolution. PhD thesis, Departement Computer Wetenschappen Web & Information Systems Engineering, Free University of Brussels. 2006.

[Pomian, 1996] Pomian J., (1996) : Mémoire d'entreprise, techniques et outils de la gestion du savoir. p. 11, Ed Sapientia.

[Pons et Keller, 1997] Pons, A. et Keller, R. K. (1997). Schema evolution in object databases by catalogs. In Proceedings of International Database Engineering and Applications Symposium (IDEAS.97), Montreal, Canada, pp. 368-376, 1997.

[Popov et al., 2004] Popov B., Kiryakov A., Ognyanoff D., Manov D. et A. Kirilov (2004). KIM – a semantic annotation platform for information extraction and retrieval. Natural Language Engineering, 10, Issues 3-4, 375-392.

[Prié et Garlatti, 2004] Prié, Y. et Garlatti, S. (2004). Méta-données et annotations dans le Web sémantique. Hors Série 2004 - Web Sémantique. Revue en Sciences du Traitement de l'Information, I3 (Information - Interaction – Intelligence), Cépaduès, Toulouse, pp. 45-68.

[Prud'hommeaux et Seaborne, 2007] Eric Prud'hommeaux et Andy Seaborne (2007). SPARQL Query Language for RDF. W3C Candidate Recommendation 14 June 2007. Download available at http://www.w3.org/TR/rdf-sparql-query/.

[Roddick, 1996] Roddick, J.F. (1996). A Survey of Schema Versioning Issues for Database Systems, Information and Software Technology, 37(7):383-393, 1996.

[Rogozan et Paquette, 2005] Rogozan, D. et Paquette, G. (2005). Managing Ontology Changes on the Semantic Web. Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence (WI'05).

[Rogozan, 2004] Rogozan, D. (2004). Gestion de l'évolution d'une ontologie : méthodes et outils pour un référencement sémantique évolutif fondé sur une analyse des changements entre versions de l'ontologie. Proposition de recherche doctorale en informatique cognitive (DIC 9410). Télé-Université du Québec.

[Sowa, 1984] Sowa J.F. (1984). Conceptual Graphs: Information Processing. in Mind and Machine. Reading, Addison Wesley.

[Spivack, 2004] Spivack, N. (2004). The Future of the Net. Disponible sur http://novaspivack.typepad.com/nova_spivacks_weblog/2004/04/new_version_of_.html

[Stojanovic et al., 2002a] Stojanovic, L., Maedche, A., Stojanovic, N. et Motik, B. (2002). User-driven ontology evolution management. Proceedings of the 13th European Conference on Knowledge Engineering and Knowledge Management (EKAW-2002), Siguenza, Spain, October 2002, pages 285-300. Springer-Verlag, 2002.

[Stojanovic et al., 2002b] Stojanovic, L., Stojanovic, N. et Handschuh, S. (2002). Evolution of the Metadata in the Ontology-based Knowledge Management Systems, The 1st German Workshop on Experience Management: Sharing Experiences about the Sharing of Experience, 2002.

[Stojanovic et al., 2003] L.Stojanovic, A.Maedche, N.Stojanovic and R.Studer. Ontology evolution as reconfiguration-design problem solving. In CAP 2003, pages 162–171. ACM, OCT 2003.

[Stojanovic, 2004] Stojanovic, L. (2004). Methods and Tools for Ontology Evolution. PhD thesis, University of Karlsruhe, 2004.

[Stuckenschmidt et Klein, 2003] H.Stuckenschmidt et M.Klein (2003). Integrity and change in modular ontologies. In Proceedings of the 18th International Joint Conference on Artificial Intelligence, Acapulco, Mexico.

[Studer et al., 1998] Studer R., Benjamins V.R. et Fensel D. (1998) Knowledge engineering: principles and methods, in IEEE Transactions on Data and Knowledge Engineering, 25(1&2), 1998, pp.161-197.

[Sunagawa et al., 2003] Sunagawa, E., Kozaki, K., Kitamura, Y. et Mizoguchi, R. (2003). An environment for distributed ontology development based on dependency management. In Proceedings of the Second International Semantic Web Conference (ISWC2003), Sanibel Island, FL, USA, pp.453-468, 2003.

[Sure et al., 2002] Sure, Y., Erdmann, M., Angele, J., Staab, S., Studer, R. et Wenke, D. (2002). OntoEdit: Collaborative Ontology Engineering for the Semantic Web. In: Horrocks I, Hendler J (eds) First International Semantic Web Conference (ISWC'02). Sardinia, Italy. Springer Verlag Lecture Notes in Computer Science (LNCS) 2342. Berlin, Germany, pp 221–235.

[Sure et Studer, 2002] Sure, Y. et Studer, R. (2002). On-To-Knowledge Methodology. In J. Davies, D. Fensel, F. van Harmelen (eds.), On-To-Knowledge: Semantic Web enabled Knowledge Management, pp. 33-46. Wiley, 2002.

[Tifous et Dieng-Kuntz, 2007] Tifous, A. et Dieng-Kuntz, R. eds (2007). CoP-dependent ontologies. Deliverable D.KNO.02 of the Palette Project FP6-028038. March 2007.

[Tifous et al., 2007] Tifous, A., El Ghali, A., Giboin, A., Dieng-Kuntz, R. (2007). O'CoP, an Ontology Dedicated to Communities of Practice, Proceedings of I-KNOW'07, The 7th International Conference on Knowledge Management, Graz, Austria, September 5-7, 2007.

[Uren et al., 2006] Uren, V., Cimiano, P., Iria, J., Handschuh, S., Vargas-Vera, M., Motta, E. et Ciravegna, F. (2006). Semantic Annotation for Knowledge Management: Requirements and a Survey of the State of the Art. Web Semantics: Science, Services and Agents on the World Wide Web 4(1):pp. 14-28.

[Uschold et Gruninger, 1996] Uschold, M. et Gruninger, M. (1996). Ontologies: Principles, methods and applications. Knowledge Engineering Review, Vol. 11:2, 93-136, 1996. Also available as AIAI-TR-191 from AIAI, The University of Edinburgh.

[Uschold et King, 1995] Uschold M. et King M., (1995). Towards a methodology for building ontologies, in Proceedings of IJCAI'95 Workshop on Basic Ontological Issues in Knowledge Sharing.

[Van Heijst et al., 1996] Van Heijst G., Van der Spek R., et Kruizinga E. (1996). Organizing Corporate Memories. In B. Gaines, M. Musen eds, Proc. of KAW'96, Banff, Canada, November, pp. 42-1 42-17.

[Van Heijst et al., 1997] Van Heijst, G., Schreiber, A. Th. et Wielinga, B. J.. (1997) Using explicit ontologies in KBS development. Int. J. of Human-Computer Studies, 46(2/3), 1997.

[Vargas-Vera et al., 2002] Vargas-Vera M., Motta E., Domingue J., Lanzoni M., Stutt A. et Ciravegna F. (2002). MnM: Ontology Driven Semi-Automatic and Automatic Support for Semantic Markup, In Proc. of the 13th International Conference on Knowledge Engineering and Knowledge Management EKAW 2002, Springer Verlag LNAI 2473, 379-391.

[Vidou et al., 2007] Vidou, G., Dieng-Kuntz, R., Ghali, A., Evangelou, C., Giboin, A., Tifous, A. et Jacquemart, S. (2007). Towards an Ontology for Knowledge Management in Communities of Practice. Proc. of 6th Int. Conf. on Practical Aspects of Knowledge Management (PAKM'2006), Vienna, Austria, 2006, Springer, LNCS 433, pp. 303-314.

[Visser et al., 1997] Visser, P. R. S., Jones, D.M., Bench-Capon, T. J. M., et Shave, M. J. R. (1997). An analysis of ontological mismatches: Heterogeneity versus interoperability. In AAAI 1997 Spring Symposium on Ontological Engineering, Stanford, USA.

[Wielinga et Schreiber, 1997] Wielinga, B. et Schreiber, G. (1997). Configuration design problem solving. IEEE Intelligent Systems, Volume 12, Number 1, pp. 49-56, March-April 1997.

[Zhang et al., 2003] Zhang, Z., Zhang, L., Lin, C.X., Zhao, Y. et Yu, Y. (2003). Data Migration for Ontology Evolution. In Poster Proceedings of the 2nd International Semantic Web Conference (ISWC-03), 2003.

Annexe A - Stratégies de résolution

<table>
<tr><td rowspan="2">1.1 InsertConcept</td><td colspan="2">Syntaxe
InsertConcept(newC, c)</td><td colspan="2">Pré-condition : c ∈ C, newC ∉ C, (newC, c) ∉ $\mathbf{H_c}$</td></tr>
<tr><td colspan="2">Sémantique
Insérer le nouveau concept newC comme fils du concept c</td><td colspan="2">Post-condition : c ∈ C, newC ∈ C, (newC, c) ∈ $\mathbf{H_c}$</td></tr>
<tr><td>a. Cas de changement</td><td colspan="2">b. Stratégie de résolution: pour les éléments dépendants dans l’ontologie</td><td colspan="2">c. Stratégie de résolution: pour les annotations concernées</td></tr>
<tr><td>c est un concept de l’ontologie

c ∈ C</td><td>SO-1</td><td>Le concept newC héritera des propriétés du concept père c et des concepts ascendants de c

Autre option (semi-automatique) : certaines instances du concept c peuvent devenir instances du type newC</td><td>SA-1</td><td>Pas de changement sur l’annotation

Autre option : certains triplets pourraient être changés selon les changements des instances pour devenir plus précis</td></tr>
</table>

Note:

- $\mathbf{H_c \subseteq C \times C}$ *: hiérarchie de concepts (relation acyclique). Si* $(c_1, c_2) \in \mathbf{H_c}$ *alors* c_1 *est un sous-concept de* c_2 *et* c_2 *est un super-concept de* c_1
- $\mathbf{H_P \subseteq P \times P}$ *: hiérarchie de propriétés (relation acyclique). Si* $(p_1, p_2) \in \mathbf{H_P}$ *alors* p_1 *est une sous-propriété de* p_2 *et* p_2 *est une super-propriété de* p_1

<table>
<tr><td rowspan="2">1.2 DeleteConcept</td><td>Syntaxe
DeleteConcept(c)</td><td>Pré-condition : c ∈ C</td></tr>
<tr><td>Sémantique
Supprimer le concept c</td><td>Post-condition : c ∉ C</td></tr>
<tr><td>a. Cas de changement</td><td>b. Stratégie de résolution: pour les éléments dépendants</td><td>c. Stratégie de résolution: pour les annotations concernées</td></tr>
</table>

	dans l'ontologie			
1.2.1 - Si c est un concept feuille, possédant un seul père c_2, et sans lien avec aucune propriété. $c, c_2 \in \mathbf{C}$ $(c, c_2) \in \mathbf{H_c}$ $\neg\exists c_1 \in \mathbf{C} : (c_1 \neq c_2) \wedge (c, c_1) \in \mathbf{H_c}$ $\neg\exists c_0 \in \mathbf{C} : (c_0, c) \in \mathbf{H_c}$ $\neg\exists p \in \mathbf{P} : c \in domain(p)$ ou $c \in range(p)$	**SO - 2**	Traiter des instances du concept c - Supprimer les instances du concept c	**SA - 2**	Traiter des triplets contenant une instance du concept c - Supprimer tous les triplets contenant une ressource du type c
		Attacher les instances du concept c au concept père c_2 de c		Remplacer le type c d'une ressource de type c par le type c_2 (i.e. concept père de c) dans les triplets concernés, i.e. remplacer (r type c) par (r type c_2)
	SO - 3	- Supprimer le lien entre le concept c et son concept père c_2 - Supprimer le concept c	**SA - 3**	Supprimer tous les triplets contenant une ressource du type c
				Remplacer le type c d'une ressource de type c par le type c_2 (i.e. concept père de c) dans les triplets concernés, i.e. remplacer (r type c) par (r type c_2)
				Autre option : Remplacer le type c *d'une ressource du type* c *par le type* c_x *indiqué par l'utilisateur*
1.2.2 - Si c est un concept feuille, possédant un seul père, avec c appartenant au domaine/co-domaine d'une seule propriété *p*. $c, c_2 \in \mathbf{C}$ $p \in \mathbf{P}$ $(c, c_2) \in \mathbf{H_c}$ $\neg\exists c_1 \in \mathbf{C} : (c_1 \neq c_2) \wedge (c, c_1) \in \mathbf{H_c}$	**SO - 4**	Traitement des instances du concept c *Idem SO - 2*	**SA - 4**	Traitement des triplets contenant une instance du type c *Idem SA – 2*
	SO - 5	- Supprimer le lien entre le concept c et son concept père c_2 - Supprimer le lien entre c et la propriété *p* - Supprimer le concept c	**SA - 5**	Si aucun ancêtre de c n'appartient au domaine (respectivement co-domaine) de la propriété *p*, alors supprimer tous les triplets contenant à la fois une ressource r du type c et la propriété *p* , i.e. les triplets (r p v) avec (r type c) ou (respectivement les triplets (r1 p r) avec (r type c))

$\neg\exists c_0 \in \mathbf{C} : (c_0, c) \in \mathbf{H_c}$ $c \in domain\ (p)$ ou $c \in range(p)$				S'il existe un ancêtre du concept c appartenant au domaine (respectivement co-domaine) de la propriété *p*, alors remplacer le type c d'une ressource r du type c apparaissant dans un triplet avec la propriété *p* par le concept père c_2 , i.e. remplacer (r type c) par (r type c_2) s'il existe un triplet (r p v) ou (respectivement (r1 p r) avec (r type c)) *Autre option : Remplacer le type* c *d'une ressource* r *du type* c *apparaissant dans un triplet avec la propriété* p *par le concept* c_x *indiqué par l'utilisateur.*
1.2.3 - Si c est un concept feuille, possédant plusieurs pères, avec c appartenant au domain/co-domaine d'une propriété *p*. $c, c_1, c_2 \in \mathbf{C}$ $p \in \mathbf{P}$ $(c, c_1) \in \mathbf{H_c}$ $(c, c_2) \in \mathbf{H_c}$ $\neg\exists c_0 \in \mathbf{C} : (c_0, c) \in \mathbf{H_c}$ $c \in domain\ (p)$ ou $c \in range(p)$	**SO - 6**	Traitement des instances du concept c *Idem SO - 2*	**SA - 6**	Traitement des triplets contenant une instance du type c *Idem SA – 2*
	SO - 7	- Supprimer le lien entre le concept c et tous ses concept pères de c (e.g. c_1, c_2…) - Supprimer le lien entre c et la propriété *p* - Supprimer le concept c	**SA - 7**	Si aucun ancêtre de c n'appartient au domaine (respectivement co-domaine) de la propriété *p*, alors supprimer tous les triplets contenant à la fois une ressource r du type c et la propriété *p* , i.e. les triplets (r p v) avec (r type c) ou (respectivement les triplets (r1 p r) avec (r type c))

				S'il existe un ancêtre du concept c appartenant au domaine (respectivement co-domaine) de la propriété *p*, alors remplacer le type c d'une ressource r du type c apparaissant dans un triplet avec la propriété *p* par le concept père c_2 , i.e. remplacer (r type c) par (r type c_2) s'il existe un triplet (r p v) ou (respectivement (r1 p r) avec (r type c))
				Si tous les ancêtres du concept c *(i.e.* c_1 *et* c_2*) appartiennent au domaine (respectivement co-domaine) de la propriété* p*, remplacer le type* c *d'une ressource* r *apparaissant dans un triplet (r p v)ou (respectivement (r1 p r)) par le concept* c_x *indiqué par l'utlisateur*
1.2.4 – Si c est un concept milieu, possédant un seul père, sans lien avec aucune propriété. $c_0, c, c_1, c_2 \in \mathbf{C}$ $(c_0, c) \in \mathbf{H_c}$ $(c, c_2) \in \mathbf{H_c}$ $\neg\exists c_1 \in \mathbf{C} : (c_1 \neq c_2) \wedge (c, c_1) \in \mathbf{H_c}$ $\neg\exists p \in \mathbf{P} : c \in domain(p)$ ou $c \in range(p)$	**SO - 8**	Traitement des instances du concept c *Idem SO - 2*	**SA - 8**	Traitement des triplets contenant une instance du type c *Idem SA – 2*
	SO – 9	- Supprimer le lien entre le concept c et son concept père c_2 - Supprimer le lien entre c et ses sous-concepts c_i - Supprimer le concept c - Supprimer les sous-concepts c_i du concept c *Note : Traiter ensuite les sous-concepts* c_i *du concept* c	**SA – 9**	Supprimer tous les triplets contenant une ressource du type c
				Remplacer le type c d'une ressource de type c par le type c_2 (i.e. concept père de c) dans les triplets concernés, i.e. remplacer (r type c) par (r type c_2)

				Autre option : Remplacer le type c *d'une ressource* r *du type* c *par le type* c_x *indiqué par l'utilisateur* *Note : Traiter ensuite les triplets contenant des ressources du type des sous-concepts* c_i *du concept* c
	SO - 10	- Supprimer le lien entre le concept c et son concept père c_2 - Supprimer le lien entre c et ses sous-concepts c_i - Supprimer le concept c - Attacher les sous-concepts c_i du concept c au concept père c_2 *Note : Traiter ensuite les sous-concepts* c_i *du concept* c	**SA - 10**	Supprimer tous les triplets contenant une ressource du type c Remplacer le type c d'une ressource de type c par le type c_2 (i.e. concept père de c) dans les triplets concernés, i.e. remplacer (r type c) par (r type c_2) *Autre option : Remplacer le type c d'une ressource r du type c par le type* c_x *indiqué par l'utilisateur* *Note : Traiter ensuite les triplets contenant des ressources du type des sous-concepts* c_i *du concept* c
1.2.5 – Si c est un concept milieu, possédant un seul père, avec c appartenant au domain/co-domaine d'une propriété *p*. $c_0, c, c_1, c_2 \in \mathbf{C}$ $p \in \mathbf{P}$ $(c_0, c) \in \mathbf{H_c}$ $(c, c_2) \in \mathbf{H_c}$ $\neg\exists c_1 \in \mathbf{C} : (c_1 \neq c_2) \wedge (c, c_1) \in \mathbf{H_c}$	**SO - 11**	Traitement des instances du concept c *Idem SO – 2*	**SA - 11**	Traitement des triplets contenant une instance du type c *Idem SA – 2*
	SO – 12	- Supprimer le lien entre le concept c et son concept père c_2 - Supprimer le lien entre c et ses sous-concepts c_i - Supprimer le lien entre c et la propriété *p* - Supprimer le concept c - Supprimer les sous-concepts c_i du concept c	**SA - 12**	Si aucun ancêtre de c n'appartient au domaine (respectivement co-domaine) de la propriété *p*, alors supprimer tous les triplets contenant à la fois une ressource r du type c et la propriété *p* , i.e. les triplets (r p v) avec (r type c) ou (respectivement les triplets (r1 p r) avec (r type c))

c ∈ domain (*p*) ou c ∈ range(*p*)		*Note : Traiter ensuite les sous-concepts* c_i *du concept* c		S'il existe un ancêtre du concept c appartenant au domaine (respectivement co-domaine) de la propriété *p*, alors remplacer le type c d'une ressource r du type c apparaissant dans un triplet avec la propriété *p* par le concept père c_2 , i.e. remplacer (r type c) par (r type c_2) s'il existe un triplet (r p v) ou (respectivement (r1 p r) avec (r type c))
				Autre option : Remplacer le type c d'une ressource r du type c apparaissant dans un triplet avec la propriété p par le concept c_x *indiqué par l'utilisateur*
	SO – 13	- Supprimer le lien entre le concept c et son concept père c_2 - Supprimer le lien entre c et ses sous-concepts c_i - Supprimer le lien entre c et la propriété *p* - Supprimer le concept c - Attacher les sous-concepts c_i du concept c au concept père c_2 *Note : Traiter ensuite les sous-concepts* c_i *du concept* c	**SA - 13**	Si aucun ancêtre de c n'appartient au domaine (respectivement co-domaine) de la propriété *p*, alors supprimer tous les triplets contenant à la fois une ressource r du type c et la propriété *p* , i.e. les triplets (r p v) avec (r type c) ou (respectivement les triplets (r1 p r) avec (r type c))

				S'il existe un ancêtre du concept c appartenant au domaine (respectivement co-domaine) de la propriété *p*, alors remplacer le type c d'une ressource r du type c apparaissant dans un triplet avec la propriété *p* par le concept père c_2 , i.e. remplacer (r type c) par (r type c_2) s'il existe un triplet (r p v) ou (respectivement (r1 p r) avec (r type c))
				Autre option : Remplacer le type c d'une ressource r du type c apparaissant dans un triplet avec la propriété p par le concept c_x indiqué par l'utilisateur *Note : Garder les triplets contenant à la fois la propriété p et une ressource de type c_i , s'il existe un lien entre un ancêtre du concept c et la propriété p. Sinon, supprimer ces triplets contenant la propriété p et une ressource de type c_i*
1.2.6 – Si c est un concept milieu, possédant plusieurs pères, avec c appartenant au domaine/co-domaine d'une propriété *p*. $c_0, c, c_1, c_2 \in \mathbf{C}, (c_1 \neq c_2)$ $p \in \mathbf{P}$ $(c_0, c) \in \mathbf{H_c}$ $(c, c_1) \in \mathbf{H_c}$	**SO - 14**	Traitement des instances du concept c *Idem SO - 2*	**SA - 14**	Traitement des triplets contenant une instance du type c *Idem SA – 2*
	SO - 15	- Supprimer le lien entre le concept c et tous ses concepts pères (e.g. c_1, c_2) - Supprimer le lien entre c et ses sous-concepts c_i - Supprimer le lien entre c et la propriété *p* - Supprimer le concept c	**SA - 15**	Si aucun ancêtre de c n'appartient au domaine (respectivement co-domaine) de la propriété *p*, alors supprimer tous les triplets contenant à la fois une ressource r du type c et la propriété *p* , i.e. les triplets (r p v) avec (r type c) ou (respectivement les triplets (r1 p r) avec (r type c))

<table>
<tr><td rowspan="3">$(c, c_2) \in \mathbf{H_c}$
$c \in$ domain (p)
ou $c \in$ range(p)</td><td></td><td>- Supprimer les sous-concepts c_i du concept c
Note : Traiter ensuite les sous-concepts c_i du concept c</td><td></td><td>S'il existe un ancêtre du concept c appartenant au domaine (respectivement co-domaine) de la propriété p, alors remplacer le type c d'une ressource r du type c apparaissant dans un triplet avec la propriété p par le concept père c_2 , i.e. remplacer (r type c) par (r type c_2) s'il existe un triplet (r p v) ou (respectivement (r1 p r) avec (r type c))</td></tr>
<tr><td></td><td></td><td>Si tous les ancêtres du concept c (i.e. c_1 et c_2) appartiennent au domaine (respectivement co-domaine) de la propriété p, remplacer le type c d'une ressource r apparaissant dans un triplet (r p v) ou (respectivement (r1 p r)) par le concept c_x indiqué par l'utlisateur</td></tr>
<tr><td>SO - 16</td><td>- Supprimer le lien entre le concept c et tous ses concepts pères (e.g. c_1, c_2)
- Supprimer le lien entre c et ses sous-concepts c_i
- Supprimer le lien entre c et la propriété p
- Supprimer le concept c
- Attacher les sous-concepts c_i du concept c au concept père quelconque c_x indiqué par l'utilisateur
Note : Traiter ensuite les sous-concepts c_i du concept c</td><td>SA - 16</td><td>Si aucun ancêtre de c n'appartient au domaine (respectivement co-domaine) de la propriété p, alors supprimer tous les triplets contenant à la fois une ressource r du type c et la propriété p , i.e. les triplets (r p v) avec (r type c) ou (respectivement les triplets (r1 p r) avec (r type c))</td></tr>
<tr><td></td><td></td><td>S'il existe un ancêtre du concept c appartenant au domaine (respectivement co-domaine) de la propriété p, alors remplacer le type c d'une ressource r du type c apparaissant dans un triplet avec la propriété p par le concept père c_2 , i.e. remplacer (r type c) par (r type c_2) s'il existe un triplet (r p v) ou (respectivement (r1 p r) avec (r type c))</td></tr>
</table>

				Si tous les ancêtres du concept c *(i.e.* c_1 *et* c_2*) appartiennent au domaine (respectivement co-domaine) de la propriété p, remplacer le type* c *d'une ressource* r *apparaissant dans un triplet* (r p v) *ou (respectivement* (r1 p r)*) par le concept* c_x *indiqué par l'utlisateur*

1.3 RenameConcept	***Syntaxe*** RenameConcept(c, nC)		***Pré-condition :*** $c \in \mathbf{C}$, name(c) $\in$ name(**C**), nC $\notin$ name(**C**) *Note : name() est la fonction qui renvoie le nom du concept*	
	Sémantique Remplacer le nom du concept c par le nouveau nom nC		***Post-condition :*** name(c) $\notin$ name(**C**), nC $\in$ name(**C**)	
a. Cas de changement	**b. Stratégie de résolution: pour les éléments dépendants dans l'ontologie**		**c. Stratégie de résolution: pour les annotations concernées**	
c est un concept de l'ontologie $c \in \mathbf{C}$	**SO - 17**	- Vérifier l'existence du nouveau nom - Remplacer le nom du concept modifié c par le nouveau nom nC	**SA - 17**	Remplacer le type c d'une ressource de type c par le type nC, i.e. remplacer (r type c) par (r type nC)

1.4 CreateHierarchyConceptLink	***Syntaxe*** CreateHierarchyConceptLink(c_1, c_2)	***Pré-condition :*** $c_1, c_2 \in \mathbf{C}$, $(c_1, c_2) \notin \mathbf{H_c}$
	Sémantique Créer un lien de subsomption entre les concepts c_1 et c_2, c_2 devient le père de c_1	***Post-condition :*** $c_1, c_2 \in \mathbf{C}$, $(c_1, c_2) \in \mathbf{H_c}$
a. Cas de changement	**b. Stratégie de résolution: pour les éléments dépendants dans l'ontologie**	**c. Stratégie de résolution: pour les annotations concernées**

c_1 et c_2 sont des concepts de l'ontologie $c_1, c_2 \in \mathbf{C}$	**SO - 18**	Le concept c_1 et ses sous-concepts hériteront les propriétés de c_2 et celles des concepts ascendantes de c_2	**SA - 18**	Pas de changement sur l'annotation

1.5 DeleteHierarchyConceptLink	***Syntaxe*** `DeleteHierarchyConceptLink(`c_1`, `c_2`)`		***Pré-condition :*** $c_1, c_2 \in \mathbf{C}, (c_1, c_2) \in \mathbf{H_c}$	
	Sémantique Supprimer le lien de subsomption entre les concepts c_1 et c_2, c_2 n'est plus le père de c_1		***Post-condition :*** $c_1, c_2 \in \mathbf{C}, (c_1, c_2) \notin \mathbf{H_c}$	
a. Cas de changement	**b. Stratégie de résolution: pour les éléments dépendants dans l'ontologie**		**c. Stratégie de résolution: pour les annotations concernées**	
c_1 et c_2 sont des concepts de l'ontologie $c_1, c_2 \in \mathbf{C}$ $p \in \mathbf{P}$	**SO – 19**	- Enlever les propriétés que le concept c_1 a héritées du concept c_2 et des super-concepts de c_2 - Supprimer le lien entre le concept c_1 et c_2	**SA – 19**	Si c_2 appartenait au domaine (respectivement co-domaine) de la propriété *p* (avant la suppression du lien entre c_1 et c_2) mais pas c_1 directement, alors supprimer tous les triplets contenant à la fois la propriété *p* et une ressource du type c_1, i.e. les triplets `(r p v)` avec `(r type `c_1`)` ou (respectivement les triplets `(r1 p r)` avec `(r type `c_1`))
				Si c_1 appartenait au domaine (respectivement co-domaine) de la propriété *p* (avant la suppression du lien entre c_1 et c_2), alors garder tous les triplets contenant à la fois la propriété *p* et une ressource du type c_1

<table>
<tr><td rowspan="2">2.1 InsertProperty</td><td colspan="2">Syntaxe
InsertProperty(p, newP)</td><td colspan="2">Pré-condition : p ∈ P, newP ∉ P, (newP, p) ∉ $\mathbf{H_P}$</td></tr>
<tr><td colspan="2">Sémantique
Insérer la nouvelle propriété newP comme fille de la propriété p</td><td colspan="2">Post-condition : p ∈ P, newP ∈ P, (newP, p) ∈ $\mathbf{H_P}$</td></tr>
<tr><td>a. Cas de changement</td><td colspan="2">b. Stratégie de résolution: pour les éléments dépendants dans l'ontologie</td><td colspan="2">c. Stratégie de résolution: pour les annotations concernées</td></tr>
<tr><td>p est une propriété de l'ontologie

p ∈ P</td><td>SO – 20</td><td>La nouvelle propriété newP héritera des concepts domaines/co-domaines de la propriété p et des propriétés ascendantes de p</td><td>SA – 20</td><td>Pas de changement sur l'annotation</td></tr>
</table>

<table>
<tr><td rowspan="2">2.2 DeleteProperty</td><td colspan="2">Syntaxe
DeleteProperty(p)</td><td colspan="2">Pré-condition : p ∈ P</td></tr>
<tr><td colspan="2">Sémantique
Supprimer la propriété p</td><td colspan="2">Post-condition : p ∉ P</td></tr>
<tr><td>a. Cas de changement</td><td colspan="2">b. Stratégie de résolution: pour les éléments dépendants dans l'ontologie</td><td colspan="2">c. Stratégie de résolution: pour les annotations concernées</td></tr>
<tr><td>2.2.1 - Si p est une propriété feuille, possédant une seule mère
p, p_2 ∈ P
(p, p_2) ∈ $\mathbf{H_P}$
¬∃p_1 ∈ P :(p_1≠p_2)∧(p, p_1) ∈ $\mathbf{H_P}$

c est un concept de l'ontologie</td><td>SO – 21</td><td>Traiter les concepts appartenant au domaine/co-domaine de p
- Supprimer tous les liens de domaine/co-domaine entre un concept et la propriété p, i.e. entre le concept c et p
- Supprimer le lien entre p et p_2
- Supprimer p</td><td>SA – 21</td><td>Traiter les triplets contenant une ressource de type c et la propriété p :
Supprimer tous les triplets contenant une ressource du type c et la propriété p, i.e. les triplets (r p v) avec (r type c)</td></tr>
</table>

	SO – 22	- Supprimer tous les liens de domaine/co-domaine entre un concept et la propriété p, i.e. entre le concept c et p - Attacher la propriété p_2 (mère de p) au concept c, selon l'indication de l'utilisateur - Supprimer p	**SA – 22**	Selon l'indication de l'utilisateur : Remplacer la propriété p par p_2 dans tous les triplets contenant une ressource du type c et la propriété p, i.e. remplacer le triplet (r p v) avec (r type c) par le triplet(r p_2 v)
2.2.2 - Si p est une propriété milieu, possédant une seule mère p_2 $p_0, p, p_2 \in \mathbf{P}$ $(p_0, p) \in \mathbf{H_P}$ $(p, p_2) \in \mathbf{H_P}$ $\neg\exists p_1 \in \mathbf{P} : (p_1 \neq p_2) \wedge (p, p_1) \in \mathbf{H_P}$ c est un concept de l'ontologie $c \in \mathbf{C}$ $c \in$ domain (p) ou $c \in$ range(p)		Traiter les concepts appartenant au domaine/co-domaine de p - Idem 2.2.1b - Selon l'indication de l'utilisateur, les sous-propriétés de p peuvent être attachées à la propriété p_2 ou être supprimées		Traiter les triplets contenant une ressource de type c et la propriété p : Idem 2.2.1c

2.3 RenameProperty	***Syntaxe*** RenameProperty(p, nP)	***Pré-condition :*** p ∈ **P**, name(p) ∈ name(**P**), nP ∉ name(**P**) *Note : name() est la fonction qui renvoie le nom de la propriété*
	Sémantique Remplacer le nom de la propriété p par le nouveau nom nP	***Post-condition :*** name(p) ∉ name(**P**), nP ∈ name(**P**)
a. Cas de changement	**b. Stratégie de résolution: pour les éléments dépendants**	**c. Stratégie de résolution: pour les annotations concernées**

		dans l'ontologie		
p est une propriété de l'ontologie $p \in \mathbf{P}$	**SO – 23**	- Vérifier l'existence du nouveau nom - Remplacer le nom de la propriété modifiée p par le nouveau nom nP	**SA – 23**	Traiter les triplets contenant la propriété p : Remplacer le nom de la propriété p par le nouveau nom nP

2.4 CreateHierarchyPropertyLink	***Syntaxe*** CreateHierarchyPropertyLink(p_1, p_2)	***Pré-condition :*** $p_1, p_2 \in \mathbf{P}, (p_1, p_2) \notin \mathbf{H_P}$
	Sémantique Créer un lien de subsomption entre les propriétés p_1 et p_2, p_2 devient la mère de p_1	***Post-condition :*** $p_1, p_2 \in \mathbf{P}, (p_1, p_2) \in \mathbf{H_P}$

a. Cas de changement		**b. Stratégie de résolution: pour les éléments dépendants dans l'ontologie**		**c. Stratégie de résolution: pour les annotations concernées**
p_1 et p_2 sont des propriétés de l'ontologie $p_1, p_2 \in \mathbf{P}$	**SO – 24**	La propriété p_1 et ses sous-propriétés hériteront les concepts domaines/co-domaines de p_2 et ceux des propriétés ascendantes de p_2	**SA – 24**	Pas de changement sur l'annotation

2.5 DeleteHierarchyPropertyLink	***Syntaxe*** DeleteHierarchyPropertyLink(p_1, p_2)	***Pré-condition :*** $p_1, p_2 \in \mathbf{P}, (p_1, p_2) \in \mathbf{H_P}$
	Sémantique Supprimer le lien de subsomption entre les propriétés p_1 et p_2, p_2 n'est plus la mère de p_1	***Post-condition :*** $p_1, p_2 \in \mathbf{P}, (p_1, p_2) \notin \mathbf{H_P}$

a. Cas de changement	**b. Stratégie de résolution: pour les éléments dépendants dans l'ontologie**	**c. Stratégie de résolution: pour les annotations concernées**

p_1 et p_2 sont des propriétés de l'ontologie $p_1, p_2 \in \mathbf{P}$ c_1 et c_2 sont des concepts de l'ontologie $c_1, c_2 \in \mathbf{C}$ c_1 a un lien direct (domaine/co-domaine) avec la propriété p_1, c_2 a un lien direct avec la propriété p_2	**SO – 25**	- Enlever des liens avec les concepts domaines/co-domaines que la propriété p_1 a hérités de la propriété p_2 et des propriétés ascendantes de p_2 - Supprimer le lien entre les propriétés p_1 et p_2	**SA – 25**	Si c_2 appartenait au domaine (respectivement co-domaine) de la propriété p_2 (avant la suppression du lien entre p_1 et p_2) mais pas de p_1 directement, alors supprimer tous les triplets contenant à la fois la propriété p_1 et une ressource du type c_2, i.e. les triplets (r p v) avec (r type c_2) ou (respectivement les triplets (r1 p r) avec (r type c_2))
				Si c_1 appartenait au domaine (respectivement co-domaine) de la propriété p_1 directement (avant la suppression du lien entre p_1 et p_2), alors garder tous les triplets contenant à la fois la propriété p_1 et une ressource du type c_1

2.6 CreatePropertyDomainLink	***Syntaxe*** CreatePropertyDomainLink (c, p)	***Pré-condition*** *:* $c \in \mathbf{C}, p \in \mathbf{P}, c \notin$ domain(p) *Note : domain(p) est la fonction qui renvoie les concepts domaines de p*
	Sémantique Créer un lien de domaine entre un concept et une propriété, c devient concept domaine de *p*	***Post-condition :*** $c \in \mathbf{C}, p \in \mathbf{P}, c \in$ domain(p)
a. Cas de changement	**b. Stratégie de résolution: pour les éléments dépendants dans l'ontologie**	**c. Stratégie de résolution: pour les annotations concernées**

c est un concept de l'ontologie c ∈ **C** *p* est une propriété de l'ontologie *p* ∈ **P**	**SO – 26**	- Créer le lien entre le concept c et la propriété *p* - Les sous-concepts de c (s'il y en a) hériteront la propriété *p* - Les sous-propriétés de *p* (s'il y en a) hériteront c comme concept domaine	**SA– 26**	Pas de changement sur l'annotation

2.7 RemovePropertyDomainLink	***Syntaxe*** `RemovePropertyDomainLink (c, p)`	***Pré-condition :*** `c ∈ C, p ∈ P, c ∈ domain(p)` *Note : domain(p) est la fonction qui renvoie les concepts domaines de p*
	Sémantique Supprimer un lien du domaine entre un concept et une propriété, c n'est plus concept domaine de *p*	***Post-condition :*** `c ∈ C, p ∈ P, c ∉ domain(p)`

a. Cas de changement		**b. Stratégie de résolution: pour les éléments dépendants dans l'ontologie**		**c. Stratégie de résolution: pour les annotations concernées**
c est un concept de l'ontologie c ∈ **C** *p* est une propriété de l'ontologie *p* ∈ **P**	**SO – 27**	- Enlever le lien entre le concept c et la propriété *p* - Les sous-concepts de c (s'il y en a) n'héritent plus la propriété *p* - Les sous-propriétés de *p* (s'il y en a) n'ont plus c comme concept domaine	**SA – 27**	Supprimer les triplets contenant des ressources du type c et la propriété *p*, i.e. les triplets `(r p v)` avec `(r type c)`

2.8 CreatePropertyRangeLink	***Syntaxe*** `CreatePropertyRangeLink (c, p)`	***Pré-condition :*** `c ∈ C, p ∈ P, c ∉ range(p)` *Note : range(p) est la fonction qui renvoie les concepts co-domaines de p*

	Sémantique Créer un lien du co-domaine entre un concept et une propriété, `c` devient concept co-domaine de *p*		***Post-condition :*** `c ∈ C, p ∈ P, c ∈ range(p)`	
a. Cas de changement	**b. Stratégie de résolution: pour les éléments dépendants dans l'ontologie**		**c. Stratégie de résolution: pour les annotations concernées**	
`c` est un concept de l'ontologie `c ∈ C` *p* est une propriété de l'ontologie *p* ∈ **P**	**SO – 28**	- Créer le lien entre le concept `c` et la propriété *p* - Les sous-concepts de `c` (s'il y en a) hériteront la propriété *p* - Les sous-propriétés de *p* (s'il y en a) hériteront `c` comme concept co-domaine	**SA – 28**	Pas de changement sur l'annotation

2.9 RemovePropertyRangeLink	***Syntaxe*** `RemovePropertyRangeLink (c, p)`		***Pré-condition :*** `c ∈ C, p ∈ P, c ∈ range(p)` *Note : range(p) est la fonction qui renvoie les concepts co-domaines de p*	
	Sémantique Supprimer un lien du co-domaine entre un concept et une propriété, `c` n'est plus concept co-domaine de *p*		***Post-condition :*** `c ∈ C, p ∈ P, c ∉ range(p)`	
a. Cas de changement	**b. Stratégie de résolution: pour les éléments dépendants dans l'ontologie**		**c. Stratégie de résolution: pour les annotations concernées**	
`c` est un concept de l'ontologie `c ∈ C` *p* est une propriété de l'ontologie *p* ∈ **P**	**SO – 29**	- Enlever le lien entre le concept `c` et la propriété *p* - Les sous-concepts de `c` (s'il y en a) n'héritent plus la propriété *p* - Les sous-propriétés de *p* (s'il y en a) n'ont plus `c` comme concept co-domaine	**SA – 29**	Supprimer les triplets contenant des ressources du type `c` et la propriété *p*, i.e. les triplets `(r1 p r)` avec `(r type c)`

<table>
<tr><td rowspan="2">3.1 CreateCommonConcept</td><td colspan="2">Syntaxe
CreateCommonConcept (c_1, c_2, newC)</td><td>Pré-condition : $c_1, c_2 \in \mathbf{C}$, newC $\notin$ C</td></tr>
<tr><td colspan="2">Sémantique
Créer un super-concept commun newC pour les concepts c_1 et c_2, et lui transférer les propriétés communes entre c_1 et c_2</td><td>Post-condition : c_1, c_2, newC $\in$ C, $(c_1$, newC$) \in \mathbf{H_c}$, $(c_2$, newC$) \in \mathbf{H_c}$</td></tr>
<tr><td>a. Cas de changement</td><td colspan="2">b. Stratégie de résolution: pour les éléments dépendants dans l'ontologie</td><td>c. Stratégie de résolution: pour les annotations concernées</td></tr>
<tr><td>c_1 et c_2 sont des concepts de l'ontologie
$c_1, c_2 \in \mathbf{C}$
p est une propriété de l'ontologie
$p \in \mathbf{P}$</td><td>SO – 30</td><td>Si c_1 et c_2 sont les concepts domaines/co-domaines d'une propriété commune p, alors cette propriété p sera attachée au concept newC et elle sera enlevée de c_1 et c_2</td><td>SA – 30</td><td>Traiter les triplets contenant les ressources du type c_1 ou c_2 et la propriété p: remplacer le nom des ressources du type c_1, c_2 par le nom du type du concept newC</td></tr>
</table>

<table>
<tr><td rowspan="2">3.2 MergeConcept</td><td colspan="2">Syntaxe
MergeConcept (c_1, c_2, newC)</td><td>Pré-condition : $c_1, c_2 \in \mathbf{C}$, newC $\notin$ C, $(c_1, c_2) \notin \mathbf{H_c}$</td></tr>
<tr><td colspan="2">Sémantique
Fusionner les concepts c_1 et c_2, et les remplaçant par un nouveau concept newC. Transférer toutes les propriétés et instances de c_1 et c_2 vers newC</td><td>Post-condition : $c_1, c_2 \notin \mathbf{C}$, newC $\in$ C</td></tr>
<tr><td>a. Cas de changement</td><td colspan="2">b. Stratégie de résolution: pour les éléments dépendants dans l'ontologie</td><td>c. Stratégie de résolution: pour les annotations concernées</td></tr>
</table>

3.2.1 - Si c_1 et c_2 sont des concepts feuilles $c_1, c_2 \in \mathbf{C}$ $\neg\exists c \in \mathbf{C} : (c, c_1) \in \mathbf{H_c}$ ou $(c, c_2) \in \mathbf{H_c}$	**SO – 31**	- Créer un nouveau concept newC - Attacher toutes les propriétés et les instances de c_1 et c_2 au nouveau concept newC - Les concepts pères de c_1 ou de c_2 deviennent concept père de newC - Supprimer c_1 et c_2	**SA – 31**	Traiter les triplets contenant les ressources du type c_1 ou c_2: Remplacer tous les triplets (r type c_1)ou (r type c_2)par(r type newC)
3.2.2 - Si c_1 et c_2 sont des concepts au milieu $c_1, c_2 \in \mathbf{C}$ $\exists c \in \mathbf{C} : (c, c_1) \in \mathbf{H_c} \wedge \exists c' \in \mathbf{C} : (c', c_2) \in \mathbf{H_c}$	**SO – 32**	Idem 3.2.1b mais on ne change pas les sous-branches de c_1 et c_2, les concepts fils de c_1 ou c_2 deviennent les concepts fils de newC	**SA – 32**	Remplacer tous les triplets (r type c_1)ou (r type c_2)par(r type newC)

3.3 SplitConcept	***Syntaxe*** SplitConcept (c, $newC_1$, $newC_2$)	***Pré-condition :*** c ∈ **C**, $newC_1$, $newC_2$ ∉ **C**, $newC_1 \neq newC_2$
	Sémantique Diviser un concept c en deux nouveaux concepts $newC_1$ et $newC_2$, et distribuer les propriétés directes (ayant un lien avec c) et les instances de c vers ces nouveaux concepts	***Post-condition :*** c ∉ **C**, $newC_1$, $newC_2$ ∈ **C**, $newC_1 \neq newC_2$
a. Cas de changement	**b. Stratégie de résolution: pour les éléments dépendants dans l'ontologie**	**c. Stratégie de résolution: pour les annotations concernées**

3.3.1 - Si c est un concept feuille, possédant un seul père $c, c_2 \in \mathbf{C}$ $(c, c_2) \in \mathbf{H_c}$ $\neg\exists c_1 \in \mathbf{C} : (c_1 \neq c_2) \wedge (c, c_1) \in \mathbf{H_c}$ $\neg\exists c_0 \in \mathbf{C} : (c_0, c) \in \mathbf{H_c}$	**SO – 33**	- Créer deux nouveaux concepts $newC_1$ et $newC_2$, attacher ces deux concepts au concept père c_2 de c - Distribuer toutes les propriétés directes et les instances de c vers $newC_1$ et $newC_2$ - Supprimer c	**SA – 33**	Traiter les triplets contenant les ressources du type c : pour chaque triplet (r p v)avec (r type c), si la propriété *p* a désormais comme domaine $newC_1$ (respectivement $newC_2$), remplacer le triplet (r type c)par (r type $newC_1$) ou (respectivement par (r type $newC_1$))
3.3.2 - Si c est un concept milieu, possédant un seul père $c_0, c, c_2 \in \mathbf{C}$ $(c_0, c) \in \mathbf{H_c}$ $(c, c_2) \in \mathbf{H_c}$ $\neg\exists c_1 \in \mathbf{C} : (c_1 \neq c_2) \wedge (c, c_1) \in \mathbf{H_c}$	**SO – 34**	- Les propriétés directes et les instances du concept c seront attachées à $newC_1$ ou à $newC_2$ selon l'indication de l'utilisateur - Attacher la sous-hiérarchie du concept c vers $newC_1$ ou $newC_2$ selon l'indication de l'utilisateur	**SA – 34**	Remplacer le type c d'une ressource du type c par $newC_1$ ou $newC_2$ selon l'indication de l'utilisateur et selon la compatibilité avec la nouvelle distribution des propriétés entre $newC_1$ et $newC_2$

3.4 InsertConceptGeneralisation	***Syntaxe*** InsertConceptGeneralisation (c, newC)	***Pré-condition*** *:* $c \in \mathbf{C}, newC \notin \mathbf{C}$
	Sémantique Insérer un nouveau concept newC entre le concept c et tous ses super-concepts directs	***Post-condition :*** $c, newC \in \mathbf{C}, (c, newC) \in \mathbf{H_c}$
a. Cas de changement	**b. Stratégie de résolution: pour les éléments dépendants dans l'ontologie**	**c. Stratégie de résolution: pour les annotations concernées**

c est un concept de l'ontologie, possédant plusieurs pères $c, c_1, c_2 \in \mathbf{C}$ $(c, c1) \in \mathbf{H_c}$ $(c, c_2) \in \mathbf{H_c}$	**SO – 35**	Pas de changement	**SA – 35**	Pas de changement

3.5 InsertConceptSpecialisation	***Syntaxe*** InsertConceptSpecialisation (c, newC)	***Pré-condition :*** $c \in \mathbf{C}$, $newC \notin \mathbf{C}$
	Sémantique Insérer un nouveau concept newC entre le concept c et tous ses sous-concepts directs	***Post-condition :*** $c, newC \in \mathbf{C}, (newC, c) \in \mathbf{H_c}$

a. Cas de changement		**b. Stratégie de résolution: pour les éléments dépendants dans l'ontologie**		**c. Stratégie de résolution: pour les annotations concernées**
c est un concept de l'ontologie, possédant plusieurs fils $c, c_1, c_2 \in \mathbf{C}$ $(c_1, c) \in \mathbf{H_c}$ $(c_2, c) \in \mathbf{H_c}$ *p* est une propriété de l'ontologie $p \in \mathbf{P}$	**SO – 36**	Si tous les sous-concepts de c (i.e. c_1 et c_2) appartiennent au domaine/co-domaine d'une même propriété *p*, alors cette propriété *p* sera attachée au concept newC	**SA – 36**	Remplacer le type des sous-concepts c_1 (ou c_2) d'une ressource du type c_1 (ou c_2) par le concept newC , i.e. remplacer les triplets (r p v)avec (r type c_1) ou (respectivement (r type c_2)) par le triplet (r type newC)

3.6 MoveConcept	***Syntaxe*** MoveConcept (c, oldC, newC)	***Pré-condition :*** $c, oldC, newC \in \mathbf{C}, (c, oldC) \in \mathbf{H_c}, (c, newC) \notin \mathbf{H_c}$

	Sémantique Déplacer le concept c qui est le fils du concept oldC à la nouvelle position comme fils du concept newC		***Post-condition :*** c, oldC, newC ∈ **C**, (c, oldC) ∉ $\mathbf{H_c}$, (c, newC) ∈ $\mathbf{H_c}$	
a. Cas de changement	**b. Stratégie de résolution: pour les éléments dépendants dans l'ontologie**		**c. Stratégie de résolution: pour les annotations concernées**	
3.6.1 - Si c est un concept feuille, avec c appartenant au domaine/co-domaine d'une propriété *p*. c ∈ **C** *p* ∈ **P** ¬∃c_0 ∈ **C** :(c_0, c) ∈ $\mathbf{H_c}$ c ∈ domain (*p*) ou c ∈ range(*p*)	**SO – 38**	- Supprimer le lien entre c et ses super-concepts - Attacher c comme fils du concept newC	**SA – 38**	Traiter les triplets contenant les ressources de type c et la propriété *p* : Si tous les concepts oldC et newC appartiennent au domaine/co-domaine de *p*, alors on ne change pas les triplets contenant à la fois la propriété *p* et la ressource de type c
				Si le concept newC n'appartient pas au domain/co-domaine de *p*, alors on supprime les triplets contenant à la fois la propriété *p* et la ressource de type c

3.6.2 – Si c est un concept milieu, avec c appartenant au domaine/co-domaine d'une propriété *p*. $c_0, c, c_1, c_2 \in \mathbf{C}$ $p \in \mathbf{P}$ $(c_0, c) \in \mathbf{H_c}$ $(c, c_2) \in \mathbf{H_c}$ $\neg\exists c_1 \in \mathbf{C} : (c_1 \neq c_2) \wedge (c, c_1) \in \mathbf{H_c}$ $c \in domain(p)$ ou $c \in range(p)$	**SO – 39**	- Supprimer le lien entre c et ses super-concepts - Attacher c et sa sous-hiérarchie au concept newC	**SA – 39**	- Idem 3.6.1c (stratégies SA-38) - Traiter ensuite les triplets contenant les types des sous-concepts de c qui appartiennent au domaine/co-domaine de la propriété *p*

4.1 CreateCommonProperty	***Syntaxe*** CreateCommonProperty (p_1, p_2, newP)		***Pré-condition*** : $p_1, p_2 \in \mathbf{P}$, newC $\notin \mathbf{P}$	
	Sémantique Créer une super-propriété commune newP pour les propriétés p_1 et p_2, et lui transférer les concepts domaines/co-domaines communs de p_1 et p_2		***Post-condition :*** p_1, p_2, newP $\in \mathbf{P}$, $(p_1$, newP$) \in \mathbf{H_P}$, $(p_2$, newP$) \in \mathbf{H_P}$	
a. Cas de changement	**b. Stratégie de résolution: pour les éléments dépendants dans l'ontologie**		**c. Stratégie de résolution: pour les annotations concernées**	
p_1 et p_2 sont des propriétés de l'ontologie $p_1, p_2 \in \mathbf{P}$ c est un concept de l'ontologie $c \in \mathbf{C}$	**SO – 40**	Si p_1 et p_2 ont un même concept domaine/co-domaine, alors ce concept sera attaché à la propriété newP et il sera enlevé de p_1 et p_2	**SA – 40**	Traiter les triplets contenant les ressources du type c et la propriété p_1 ou p_2 : remplacer la propriété p_1 ou p_2 par newP

<table>
<tr><td rowspan="2">4.2 MergeProperty</td><td colspan="2">Syntaxe
MergeProperty (p_1, p_2, newP)</td><td>Pré-condition : p_1, p_2 $\in$ P, newP $\notin$ P, (p_1, p_2) $\notin$ $\mathbf{H_P}$</td></tr>
<tr><td colspan="2">Sémantique
Fusionner les propriétés p_1 et p_2, et les remplaçant par une nouvelle propriété newP. Transférer les domaines/co-domaines p_1 et p_2 vers newP</td><td>Post-condition : p_1, p_2 $\notin$ P, newP $\in$ P</td></tr>
<tr><td>a. Cas de changement</td><td colspan="2">b. Stratégie de résolution: pour les éléments dépendants dans l'ontologie</td><td>c. Stratégie de résolution: pour les annotations concernées</td></tr>
<tr><td>p_1 et p_2 sont les propriétés feuilles
p_1, p_2 $\in$ P
$\neg\exists p \in$ P :(p, p_1) $\in$ $\mathbf{H_P}$ ou(p, p_2) $\in$ $\mathbf{H_P}$</td><td>SO – 41</td><td>- Créer une nouvelle propriété newP
- Attacher tous les concepts domaines/co-domaines p_1 et p_2 à la nouvelle propriété newP
- Supprimer p_1 et p_2</td><td>SA – 41</td><td>Traiter les triplets contenant la propriété p_1 ou p_2: remplacer la propriété p_1 ou p_2 par la nouvelle propriété newP</td></tr>
</table>

<table>
<tr><td rowspan="2">4.3 SplitProperty</td><td>Syntaxe
SplitProperty (p, $newP_1$, $newP_2$)</td><td>Pré-condition : p $\in$ P, $newP_1$, $newP_2$ $\notin$ P, (p_1, p_2) $\notin$ $\mathbf{H_P}$</td></tr>
<tr><td>Sémantique
Diviser une propriété p en deux nouvelles propriétés $newP_1$ et $newP_2$, et distribuer les concepts domaines/co-domaines de p entre ces nouvelles propriétés.</td><td>Post-condition : p $\notin$ P, $newP_1$, $newP_2$ $\in$ P, $newP_1 \neq newP_2$</td></tr>
</table>

a. Cas de changement	**b. Stratégie de résolution: pour les éléments dépendants dans l'ontologie**		**c. Stratégie de résolution: pour les annotations concernées**	
p est une propriété feuille $p, p_2 \in \mathbf{P}$ $(p, p_2) \in \mathbf{H_P}$ $\neg\exists p_0 \in \mathbf{P} : (p \neq p_0) \wedge (p_0, p) \in \mathbf{H_P}$	**SO – 42**	- Créer deux nouvelles propriétés newP_1 et newP_2, attacher ces deux propriétés à la propriété mère p_2 de p - Distribuer les concepts domaines/co-domaines de p entre newP_1 et newP_2 - Supprimer p	**SA – 42**	Traiter les triplets contenant la propriété p : remplacer chaque triplet contenant la propriété p par deux triplets dans lesquels remplacer p par newP_1 et newP_2 respectivement

4.4 InsertPropertyGeneralisation	***Syntaxe*** `InsertPropertyGeneralisation (p, newP)`	***Pré-condition*** *:* $p \in \mathbf{P}, \text{newP} \notin \mathbf{P}$
	Sémantique Insérer une nouvelle propriété newP entre la propriété p et toutes ses super-propriétés directes	***Post-condition :*** $p, \text{newP} \in \mathbf{P}, (p, \text{newP}) \in \mathbf{H_P}$

a. Cas de changement	**b. Stratégie de résolution: pour les éléments dépendants dans l'ontologie**		**c. Stratégie de résolution: pour les annotations concernées**	
p est une propriété de l'ontologie, possédant plusieurs mères $p, p_1, p_2 \in \mathbf{P}$ $(p, p_1) \in \mathbf{H_P}$ $(p, p_2) \in \mathbf{H_P}$	**SO – 43**	Pas de changement	**SA – 43**	Pas de changement

<table>
<tr><td rowspan="2">4.5 InsertPropertySpecialisation</td><td>Syntaxe
InsertPropertySpecialisation (p, newP)</td><td>Pré-condition : p ∈ P, newP ∉ P</td></tr>
<tr><td>Sémantique
Insérer une nouvelle propriété newP entre la propriété p et toutes ses sous-propriétés directes</td><td>Post-condition : p, newP ∈ P, (newP, p) ∈ $\mathbf{H_P}$</td></tr>
<tr><td>a. Cas de changement</td><td colspan="2">b. Stratégie de résolution: pour les éléments dépendants dans l'ontologie</td><td colspan="2">c. Stratégie de résolution: pour les annotations concernées</td></tr>
<tr><td>p est une propriété de l'ontologie, possédant plusieurs filles
$p, p_1, p_2 \in \mathbf{P}$
$(p_1, p) \in \mathbf{H_P}$
$(p_2, p) \in \mathbf{H_P}$</td><td>SO – 44</td><td>Pas de changement</td><td>SA – 44</td><td>Pas de changement</td></tr>
</table>

<table>
<tr><td rowspan="2">4.6 MoveProperty</td><td>Syntaxe
MoveProperty (p, oldP, newP)</td><td>Pré-condition : p, oldP, newP ∈ P, (p, oldP) ∈ $\mathbf{H_P}$, (p, newP) ∉ $\mathbf{H_P}$</td></tr>
<tr><td>Sémantique
Déplacer la propriété p qui est la sous-propriété de oldP à la nouvelle position comme la sous-propriété de newP</td><td>Post-condition : p, oldP, newP ∈ P, (p, oldP) ∉ $\mathbf{H_P}$, (p, newP) ∈ $\mathbf{H_P}$</td></tr>
<tr><td>a. Cas de changement</td><td>b. Stratégie de résolution: pour les éléments dépendants dans l'ontologie</td><td>c. Stratégie de résolution: pour les annotations concernées</td></tr>
</table>

p est une propriété feuille, avec c appartenant au domaine/co-domaine de p $p, p_2 \in \mathbf{P}$ $(p, p_2) \in \mathbf{H_P}$ $\neg\exists p_0 \in \mathbf{P} : (p \neq p_0) \wedge (p_0, p) \in \mathbf{H_P}$ $c \in \mathbf{C}$ $c \in$ domain (p) ou $c \in$ range(p)	**SO – 45**	- Supprimer le lien entre p et ses super-propriétés - Attacher p comme fille de la propriété newP	**SA – 45**	Traiter les triplets contenant les ressources du type c et la propriété p: Si le concept c appartient au domaine/co-domaine de oldP et de newP , alors on ne change pas les triplets contenant à la fois la propriété p et la ressource de type c Si le concept c n'appartient pas au domaine/co-domaine de newP, alors on supprime les triplets contenant à la fois la propriété p et la ressource de type c

Annexe B – Ontologie d'évolution CoSWEM

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
       xmlns:ev="http://www.inria.fr/acacia/coswem#"
       xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
       xmlns:cos="http://www.inria.fr/acacia/corese#"
       xmlns:owl="http://www.w3.org/2002/07/owl#"
       xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
       xml:base  ="http://www.inria.fr/acacia/coswem#">


<!-- Ontology Information -->
<owl:Ontology rdf:about="http://www.iria.fr/acacia/CoSWEM-onto.rdfs">
       <rdfs:label xml:lang="en">EvolutionOntology - Created by Hiep</rdfs:label>
       <rdfs:label xml:lang="fr">OntologieEvolution - Créée par Hiep</rdfs:label>
       <rdfs:comment xml:lang="en">EvolutionOntology describes ontology
changes</rdfs:comment>
       <rdfs:comment xml:lang="fr">OntologieEvolution décrit les changements
ontologiques</rdfs:comment>
       <owl:versionInfo xml:lang="fr">Version 1.0 - 2007-04-22</owl:versionInfo>
</owl:Ontology>

<!-- Description for Trace branch -->
<rdfs:Class rdf:ID="Trace">
       <rdfs:comment xml:lang="en">A trace of changes</rdfs:comment>
       <rdfs:comment xml:lang="fr">Une trace des changements</rdfs:comment>
       <rdfs:label xml:lang="en">a trace</rdfs:label>
       <rdfs:label xml:lang="fr">une trace</rdfs:label>
</rdfs:Class>

<!-- Description for TraceAnnot branch -->
<rdfs:Class rdf:ID="TraceAnnot">
       <rdfs:subClassOf rdf:resource="#Trace"/>
       <rdfs:comment xml:lang="en">A trace of annotation changes</rdfs:comment>
       <rdfs:comment xml:lang="fr">Une trace des changements de
l'annotation</rdfs:comment>
       <rdfs:label xml:lang="en">an annotation trace</rdfs:label>
       <rdfs:label xml:lang="fr">une trace d'annotation</rdfs:label>
</rdfs:Class>

<!-- Description for TraceOnto branch -->
<rdfs:Class rdf:ID="TraceOnto">
       <rdfs:subClassOf rdf:resource="#Trace"/>
       <rdfs:comment xml:lang="en">A trace of ontology changes</rdfs:comment>
       <rdfs:comment xml:lang="fr">Une trace des changements ontologiques</rdfs:comment>
       <rdfs:label xml:lang="en">an ontology trace</rdfs:label>
       <rdfs:label xml:lang="fr">une trace d'ontologie</rdfs:label>
</rdfs:Class>

<rdf:Property rdf:ID="hasNumber">
       <rdfs:domain rdf:resource="#TraceOnto"/>
       <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#token"/>
       <rdfs:comment xml:lang="en">TraceOnto has a number</rdfs:comment>
       <rdfs:comment xml:lang="fr">TraceOnto a un numero</rdfs:comment>
       <rdfs:label xml:lang="en">has a number</rdfs:label>
       <rdfs:label xml:lang="fr">avoir un numero</rdfs:label>
</rdf:Property>

<rdf:Property rdf:ID="hasDate">
       <rdfs:domain rdf:resource="#TraceOnto"/>
       <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#date"/>
       <rdfs:comment xml:lang="en">TraceOnto has a date</rdfs:comment>
       <rdfs:comment xml:lang="fr">TraceOnto a une date</rdfs:comment>
       <rdfs:label xml:lang="en">has a date</rdfs:label>
       <rdfs:label xml:lang="fr">avoir une date</rdfs:label>
</rdf:Property>

<rdf:Property rdf:ID="hasAuthor">
       <rdfs:domain rdf:resource="#TraceOnto"/>
       <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#token"/>
       <rdfs:comment xml:lang="en">author of ontology change</rdfs:comment>
```

```
        <rdfs:comment xml:lang="fr">auteur des changements de l'ontologie</rdfs:comment>
        <rdfs:label xml:lang="en">has an author</rdfs:label>
        <rdfs:label xml:lang="fr">avoir un auteur</rdfs:label>
</rdf:Property>

<rdf:Property rdf:ID="hasVersionBefore">
        <rdfs:domain rdf:resource="#TraceOnto"/>
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#token"/>
        <rdfs:comment xml:lang="en">TraceOnto has a version before</rdfs:comment>
        <rdfs:comment xml:lang="fr">TraceOnto a une version avant</rdfs:comment>
        <rdfs:label xml:lang="en">has a version before</rdfs:label>
        <rdfs:label xml:lang="fr">avoir une version avant</rdfs:label>
</rdf:Property>

<rdf:Property rdf:ID="hasVersionAfter">
        <rdfs:domain rdf:resource="#TraceOnto"/>
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#token"/>
        <rdfs:comment xml:lang="en">TraceOnto has a version after</rdfs:comment>
        <rdfs:comment xml:lang="fr">TraceOnto a une version apres</rdfs:comment>
        <rdfs:label xml:lang="en">has a version after</rdfs:label>
        <rdfs:label xml:lang="fr">avoir une version apres</rdfs:label>
</rdf:Property>

<rdf:Property rdf:ID="hasTrace">
        <rdfs:domain rdf:resource="#TraceOnto"/>
        <rdfs:range rdf:resource="#Change"/>
        <rdfs:comment xml:lang="en">each TraceOnto has some traces including ontology
changes</rdfs:comment>
        <rdfs:comment xml:lang="fr">chaque TraceOnto a certaines traces contenant des
changements ontologiques</rdfs:comment>
        <rdfs:label xml:lang="en">has a trace</rdfs:label>
        <rdfs:label xml:lang="fr">avoir une trace</rdfs:label>
</rdf:Property>

<!-- Description for Change branch -->
<rdfs:Class rdf:ID="Change">
        <rdfs:comment xml:lang="en">A ontology change</rdfs:comment>
        <rdfs:comment xml:lang="fr">Un changement ontologique</rdfs:comment>
        <rdfs:label xml:lang="en">a change</rdfs:label>
        <rdfs:label xml:lang="fr">un changement</rdfs:label>
</rdfs:Class>

<rdf:Property rdf:ID="hasCause">
        <rdfs:domain rdf:resource="#Change"/>
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#token"/>
        <rdfs:comment xml:lang="en">cause of change</rdfs:comment>
        <rdfs:comment xml:lang="fr">cause de changement</rdfs:comment>
        <rdfs:label xml:lang="en">has a cause</rdfs:label>
        <rdfs:label xml:lang="fr">avoir une cause</rdfs:label>
</rdf:Property>

<rdf:Property rdf:ID="hasChange">
        <rdfs:domain rdf:resource="#Change"/>
        <rdfs:range rdf:resource="#Change"/>
        <rdfs:comment xml:lang="en">each change may have another change</rdfs:comment>
        <rdfs:comment xml:lang="fr">chaque changement peut avoir un autre
changement</rdfs:comment>
        <rdfs:label xml:lang="en">has a change</rdfs:label>
        <rdfs:label xml:lang="fr">avoir un changement</rdfs:label>
</rdf:Property>

<rdf:Property rdf:ID="hasAdditionalChange">
        <rdfs:domain rdf:resource="#Change"/>
        <rdfs:range rdf:resource="#Change"/>
        <rdfs:comment xml:lang="en">each change may have some additional
changes</rdfs:comment>
        <rdfs:comment xml:lang="fr">chaque changement peut avoir certains changements
supplémentaires</rdfs:comment>
        <rdfs:label xml:lang="en">has an additional change</rdfs:label>
        <rdfs:label xml:lang="fr">avoir un changement supplémentaire</rdfs:label>
</rdf:Property>

<rdf:Property rdf:ID="hasAnnotInconsistency">
        <rdfs:domain rdf:resource="#Change"/>
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#token"/>
        <rdfs:comment xml:lang="en">each change may impact to the annotation
consistency</rdfs:comment>
        <rdfs:comment xml:lang="fr">chaque changement pourrait influencer la consistance
de l'annotation</rdfs:comment>
        <rdfs:label xml:lang="en">has an annotation inconsistency</rdfs:label>
        <rdfs:label xml:lang="fr">avoir une inconsistance d'annotation</rdfs:label>
</rdf:Property>

<!-- Ontology changes are classified into Elementary and Composite changes-->
<rdfs:Class rdf:ID="ElementaryChange">
```

```
        <rdfs:subClassOf rdf:resource="#Change"/>
        <rdfs:comment xml:lang="en">An elementary ontology change</rdfs:comment>
        <rdfs:comment xml:lang="fr">Un changement ontologique elementaire</rdfs:comment>
        <rdfs:label xml:lang="en">an elementary change</rdfs:label>
        <rdfs:label xml:lang="fr">un changement elementaire</rdfs:label>
</rdfs:Class>

<rdfs:Class rdf:ID="CompositeChange">
        <rdfs:subClassOf rdf:resource="#Change"/>
        <rdfs:comment xml:lang="en">A composite ontology change</rdfs:comment>
        <rdfs:comment xml:lang="fr">Un changement ontologique composite</rdfs:comment>
        <rdfs:label xml:lang="en">a composite change</rdfs:label>
        <rdfs:label xml:lang="fr">un changement composite</rdfs:label>
</rdfs:Class>

<!-- Elementary change is classified by Concept, Property change -->
<!-- 1.Description for Elementary ConceptChange branch -->
<rdfs:Class rdf:ID="E_ConceptChange">
        <rdfs:subClassOf rdf:resource="#ElementaryChange"/>
        <rdfs:comment xml:lang="en">An elementary ontology change for
concept</rdfs:comment>
        <rdfs:comment xml:lang="fr">Un changement ontologique elementaire pour le
concept</rdfs:comment>
        <rdfs:label xml:lang="en">an elementary change on concept</rdfs:label>
        <rdfs:label xml:lang="fr">un changement élémentaire sur le concept</rdfs:label>
</rdfs:Class>

<!-- 1.1.InsertConcept [param1:hasInsertConcept] to [param2:hasInsertConceptReference] -
->
<rdfs:Class rdf:ID="InsertConcept">
        <rdfs:subClassOf rdf:resource="#E_ConceptChange"/>
        <rdfs:comment xml:lang="en">insert a new concept into ontology</rdfs:comment>
        <rdfs:comment xml:lang="fr">insérer un nouveau concept à
l'ontologie</rdfs:comment>
        <rdfs:label xml:lang="en">insert a concept</rdfs:label>
        <rdfs:label xml:lang="fr">insérer un concept</rdfs:label>
</rdfs:Class>

<rdf:Property rdf:ID="hasInsertConcept">
        <rdfs:domain rdf:resource="#InsertConcept"/>
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#token"/>
        <rdfs:comment xml:lang="en">name of concept to be inserted</rdfs:comment>
        <rdfs:comment xml:lang="fr">nom du concept qui sera insére</rdfs:comment>
        <rdfs:label xml:lang="en">has a name inserted concept</rdfs:label>
        <rdfs:label xml:lang="fr">avoir un nom du concept à insérer</rdfs:label>
</rdf:Property>

<rdf:Property rdf:ID="hasInsertConceptReference">
        <rdfs:domain rdf:resource="#InsertConcept"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Class"/>
        <rdfs:comment xml:lang="en">the inserted concept will be child of this referent
concept</rdfs:comment>
        <rdfs:comment xml:lang="fr">le concept inséré est le fils du concept référent
</rdfs:comment>
        <rdfs:label xml:lang="en">has a concept reference</rdfs:label>
        <rdfs:label xml:lang="fr">avoir un concept référent</rdfs:label>
</rdf:Property>

<!-- 1.2.DeleteConcept [param1:hasDeleteConcept] -->
<rdfs:Class rdf:ID="DeleteConcept">
        <rdfs:subClassOf rdf:resource="#E_ConceptChange"/>
        <rdfs:comment xml:lang="en">remove a concept from ontology</rdfs:comment>
        <rdfs:comment xml:lang="fr">enlever un concept de l'ontologie</rdfs:comment>
        <rdfs:label xml:lang="en">delete a concept</rdfs:label>
        <rdfs:label xml:lang="fr">supprimer un concept</rdfs:label>
</rdfs:Class>

<rdf:Property rdf:ID="hasDeleteConcept">
        <rdfs:domain rdf:resource="#DeleteConcept"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Class"/>
        <rdfs:comment xml:lang="en">name of concept to be deleted</rdfs:comment>
        <rdfs:comment xml:lang="fr">nom du concept qui sera supprimé</rdfs:comment>
        <rdfs:label xml:lang="en">has a deleted concept</rdfs:label>
        <rdfs:label xml:lang="fr">avoir un concept à supprimer</rdfs:label>
</rdf:Property>

<!-- 1.3.RenameConcept [param1:hasOldConcept] to [param2:hasNewConceptName] -->
<rdfs:Class rdf:ID="RenameConcept">
        <rdfs:subClassOf rdf:resource="#E_ConceptChange"/>
        <rdfs:comment xml:lang="en">rename a concept of ontology</rdfs:comment>
        <rdfs:comment xml:lang="fr">renommer un concept de l'ontologie</rdfs:comment>
        <rdfs:label xml:lang="en">rename a concept</rdfs:label>
        <rdfs:label xml:lang="fr">renommer un concept</rdfs:label>
</rdfs:Class>
```

```
<rdf:Property rdf:ID="hasOldConcept">
        <rdfs:domain rdf:resource="#RenameConcept"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Class"/>
        <rdfs:comment xml:lang="en">old concept to be renamed</rdfs:comment>
        <rdfs:comment xml:lang="fr">ancien concept à renommer</rdfs:comment>
        <rdfs:label xml:lang="en">has old concept</rdfs:label>
        <rdfs:label xml:lang="fr">avoir ancien  concept</rdfs:label>
</rdf:Property>

<rdf:Property rdf:ID="hasNewConceptName">
        <rdfs:domain rdf:resource="#RenameConcept"/>
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#token"/>
        <rdfs:comment xml:lang="en">new name of concept</rdfs:comment>
        <rdfs:comment xml:lang="fr">nouveau nom du concept renommé</rdfs:comment>
        <rdfs:label xml:lang="en">has a new concept name</rdfs:label>
        <rdfs:label xml:lang="fr">avoir un nouveau nom</rdfs:label>
</rdf:Property>

<!-- 1.4.CreateHierarchyConceptLink [param1:hasSubConcept] [param2:hasSuperConcept] -->
<rdfs:Class rdf:ID="CreateHierarchyConceptLink">
        <rdfs:subClassOf rdf:resource="#E_ConceptChange"/>
        <rdfs:comment xml:lang="en">create a hierarchy link between 2 concepts of
ontology</rdfs:comment>
        <rdfs:comment xml:lang="fr">créer un lien hierarchique entre 2 concepts de
l'ontologie</rdfs:comment>
        <rdfs:label xml:lang="en">create a hierarchy concept link</rdfs:label>
        <rdfs:label xml:lang="fr">créer un lien hierarchique des concepts</rdfs:label>
</rdfs:Class>

<rdf:Property rdf:ID="hasSubConcept">
        <rdfs:domain rdf:resource="#CreateHierarchyConceptLink"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Class"/>
        <rdfs:comment xml:lang="en">indicate the sub-concept involved in the hierarchy
link created</rdfs:comment>
        <rdfs:comment xml:lang="fr">indique le sous-concept dans le lien hiérarchique
crée</rdfs:comment>
        <rdfs:label xml:lang="en">has a sub-concept</rdfs:label>
        <rdfs:label xml:lang="fr">avoir un sous-concept</rdfs:label>
</rdf:Property>

<rdf:Property rdf:ID="hasSuperConcept">
        <rdfs:domain rdf:resource="#CreateHierarchyConceptLink"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Class"/>
        <rdfs:comment xml:lang="en">indicate the super-concept involved in the hierarchy
link created</rdfs:comment>
        <rdfs:comment xml:lang="fr">indique le super-concept dans le lien hiérarchique
crée</rdfs:comment>
        <rdfs:label xml:lang="en">has a super-concept</rdfs:label>
        <rdfs:label xml:lang="fr">avoir un super-concept</rdfs:label>
</rdf:Property>

<!-- 1.5.RemoveHierarchyConceptLink [param1:hasSubConceptRemove]
[param2:hasSuperConceptRemove] -->
<rdfs:Class rdf:ID="RemoveHierarchyConceptLink">
        <rdfs:subClassOf rdf:resource="#E_ConceptChange"/>
        <rdfs:comment xml:lang="en">remove a hierarchy link between 2 concepts of
ontology</rdfs:comment>
        <rdfs:comment xml:lang="fr">supprimer un lien hierarchique entre 2 concepts de
l'ontologie</rdfs:comment>
        <rdfs:label xml:lang="en">remove a hierarchy concept link</rdfs:label>
        <rdfs:label xml:lang="fr">supprimer un lien hierarchique entre des
concepts</rdfs:label>
</rdfs:Class>

<rdf:Property rdf:ID="hasSubConceptRemove">
        <rdfs:domain rdf:resource="#RemoveHierarchyConceptLink"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Class"/>
        <rdfs:comment xml:lang="en">indicate the sub-concept involved in the hierarchy
link removed</rdfs:comment>
        <rdfs:comment xml:lang="fr">indique le sous-concept dans le lien hiérarchique
supprimé</rdfs:comment>
        <rdfs:label xml:lang="en">has a sub-concept</rdfs:label>
        <rdfs:label xml:lang="fr">avoir un sous-concept</rdfs:label>
</rdf:Property>

<rdf:Property rdf:ID="hasSuperConceptRemove">
        <rdfs:domain rdf:resource="#RemoveHierarchyConceptLink"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Class"/>
        <rdfs:comment xml:lang="en">indicate the super-concept involved in the hierarchy
link removed</rdfs:comment>
        <rdfs:comment xml:lang="fr">indique le super-concept dans le lien hiérarchique
supprimé</rdfs:comment>
        <rdfs:label xml:lang="en">has a super-concept</rdfs:label>
        <rdfs:label xml:lang="fr">avoir un super-concept</rdfs:label>
</rdf:Property>
```

```
<!-- 2. Description for Elementary PropertyChange branch -->

<rdfs:Class rdf:ID="E_PropertyChange">
        <rdfs:subClassOf rdf:resource="#ElementaryChange"/>
        <rdfs:comment xml:lang="en">An elementary ontology change for
property</rdfs:comment>
        <rdfs:comment xml:lang="fr">Un changement ontologique elementaire pour la
propriété</rdfs:comment>
        <rdfs:label xml:lang="en">an elementary change of property</rdfs:label>
        <rdfs:label xml:lang="fr">un changement élémentaire de propriété</rdfs:label>
</rdfs:Class>

<!-- 2.1.InsertProperty [param1:hasInsertProperty] to
[param2:hasInsertPropertyReference] -->
<rdfs:Class rdf:ID="InsertProperty">
        <rdfs:subClassOf rdf:resource="#E_PropertyChange"/>
        <rdfs:comment xml:lang="en">insert a property into ontology</rdfs:comment>
        <rdfs:comment xml:lang="fr">insérer une propriéte à l'ontologie</rdfs:comment>
        <rdfs:label xml:lang="en">insert a property</rdfs:label>
        <rdfs:label xml:lang="fr">insérer une propriété</rdfs:label>
</rdfs:Class>

<rdf:Property rdf:ID="hasInsertProperty">
        <rdfs:domain rdf:resource="#InsertProperty"/>
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#token"/>
        <rdfs:comment xml:lang="en">name of property to be inserted</rdfs:comment>
        <rdfs:comment xml:lang="fr">nom de la propriété qui sera inserée</rdfs:comment>
        <rdfs:label xml:lang="en">has an inserted property name</rdfs:label>
        <rdfs:label xml:lang="fr">avoir un nom de propriété à insérer</rdfs:label>
</rdf:Property>

<rdf:Property rdf:ID="hasInsertPropertyReference">
        <rdfs:domain rdf:resource="#InsertProperty"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
        <rdfs:comment xml:lang="en">the inserted property will be child of this referent
property</rdfs:comment>
        <rdfs:comment xml:lang="fr">la propriété insérée sera la fille de la propriété
référente</rdfs:comment>
        <rdfs:label xml:lang="en">has an property reference</rdfs:label>
        <rdfs:label xml:lang="fr">avoir une propriété référente</rdfs:label>
</rdf:Property>

<!-- 2.2.DeleteProperty [param1:hasDeleteProperty] -->
<rdfs:Class rdf:ID="DeleteProperty">
        <rdfs:subClassOf rdf:resource="#E_PropertyChange"/>
        <rdfs:comment xml:lang="en">remove a property from ontology</rdfs:comment>
        <rdfs:comment xml:lang="fr">enlever une propriété de l'ontologie</rdfs:comment>
        <rdfs:label xml:lang="en">delete a property</rdfs:label>
        <rdfs:label xml:lang="fr">supprimer une propriété</rdfs:label>
</rdfs:Class>

<rdf:Property rdf:ID="hasDeleteProperty">
        <rdfs:domain rdf:resource="#DeleteProperty"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
        <rdfs:comment xml:lang="en">the property to be deleted</rdfs:comment>
        <rdfs:comment xml:lang="fr">la propriété qui sera supprimée</rdfs:comment>
        <rdfs:label xml:lang="en">has a delete property</rdfs:label>
        <rdfs:label xml:lang="fr">avoir une propriété à supprimer</rdfs:label>
</rdf:Property>

<!-- 2.3.RenameProperty [param1:hasOldProperty] to [param2:hasNewPropertyName] -->
<rdfs:Class rdf:ID="RenameProperty">
        <rdfs:subClassOf rdf:resource="#E_PropertyChange"/>
        <rdfs:comment xml:lang="en">rename a property of ontology</rdfs:comment>
        <rdfs:comment xml:lang="fr">renommer une propriété de l'ontologie</rdfs:comment>
        <rdfs:label xml:lang="en">rename a property</rdfs:label>
        <rdfs:label xml:lang="fr">renommer une propriété</rdfs:label>
</rdfs:Class>

<rdf:Property rdf:ID="hasOldProperty">
        <rdfs:domain rdf:resource="#RenameProperty"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
        <rdfs:comment xml:lang="en">old property</rdfs:comment>
        <rdfs:comment xml:lang="fr">ancienne propriété</rdfs:comment>
        <rdfs:label xml:lang="en">has old property</rdfs:label>
        <rdfs:label xml:lang="fr">avoir pour ancienne propriété</rdfs:label>
</rdf:Property>

<rdf:Property rdf:ID="hasNewPropertyName">
        <rdfs:domain rdf:resource="#RenameProperty"/>
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#token"/>
        <rdfs:comment xml:lang="en">new name of property</rdfs:comment>
        <rdfs:comment xml:lang="fr">nouveau nom de la propriété</rdfs:comment>
        <rdfs:label xml:lang="en">has a new property name</rdfs:label>
```

```
        <rdfs:label xml:lang="fr">avoir nouveau nom de propriété</rdfs:label>
</rdf:Property>

<!-- 2.4.CreateHierarchyPropertyLink [param1:hasSubPropertyCreate]
[param2:hasSuperPropertyCreate] -->
<rdfs:Class rdf:ID="CreateHierarchyPropertyLink">
        <rdfs:subClassOf rdf:resource="#E_PropertyChange"/>
        <rdfs:comment xml:lang="en">create a hierarchy link between 2 properties of
ontology</rdfs:comment>
        <rdfs:comment xml:lang="fr">créer un lien hierarchique entre 2 propriétés de
l'ontologie</rdfs:comment>
        <rdfs:label xml:lang="en">create a hierarchy property link</rdfs:label>
        <rdfs:label xml:lang="fr">créer un lien hiérarchique de propriété</rdfs:label>
</rdfs:Class>

<rdf:Property rdf:ID="hasSubPropertyCreate">
        <rdfs:domain rdf:resource="#CreateHierarchyPropertyLink"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
        <rdfs:comment xml:lang="en">indicate the sub-property involved in the hierarchy
link created</rdfs:comment>
        <rdfs:comment xml:lang="fr">indique le sous-propriété dans le lien hiérarchique
créé</rdfs:comment>
        <rdfs:label xml:lang="en">has a sub-property to be linked</rdfs:label>
        <rdfs:label xml:lang="fr">avoir une sous-propriété à lier</rdfs:label>
</rdf:Property>

<rdf:Property rdf:ID="hasSuperPropertyCreate">
        <rdfs:domain rdf:resource="#CreateHierarchyPropertyLink"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
        <rdfs:comment xml:lang="en">indicate the super-property involved in the hierarchy
link created</rdfs:comment>
        <rdfs:comment xml:lang="fr">indique le super-propriété dans le lien hiérarchique
créé</rdfs:comment>
        <rdfs:label xml:lang="en">has a super-property to be linked</rdfs:label>
        <rdfs:label xml:lang="fr">avoir une super-propriété à lier</rdfs:label>
</rdf:Property>

<!-- 2.5.RemoveHierarchyPropertyLink [param1:hasSubPropertyRemove]
[param2:hasSuperPropertyRemove] -->
<rdfs:Class rdf:ID="RemoveHierarchyPropertyLink">
        <rdfs:subClassOf rdf:resource="#E_PropertyChange"/>
        <rdfs:comment xml:lang="en">remove a hierarchy link between 2 properties of
ontology</rdfs:comment>
        <rdfs:comment xml:lang="fr">enlever un lien hierarchique entre 2 propriétés de
l'ontologie</rdfs:comment>
        <rdfs:label xml:lang="en">remove a hierarchy property link</rdfs:label>
        <rdfs:label xml:lang="fr">enlever un lien hiérarchique de propriété</rdfs:label>
</rdfs:Class>

<rdf:Property rdf:ID="hasSubPropertyRemove">
        <rdfs:domain rdf:resource="#RemoveHierarchyPropertyLink"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
        <rdfs:comment xml:lang="en">indicate the sub-property involved in the hierarchy
link removed</rdfs:comment>
        <rdfs:comment xml:lang="fr">indique le sous-propriété dans le lien hiérarchique
supprimée</rdfs:comment>
        <rdfs:label xml:lang="en">has a sub-property</rdfs:label>
        <rdfs:label xml:lang="fr">avoir une sous-propriété</rdfs:label>
</rdf:Property>

<rdf:Property rdf:ID="hasSuperPropertyRemove">
        <rdfs:domain rdf:resource="#RemoveHierarchyPropertyLink"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
        <rdfs:comment xml:lang="en">indicate the super-property involved in the hierarchy
link removed</rdfs:comment>
        <rdfs:comment xml:lang="fr">indique le super-propriété dans le lien hiérarchique
supprimée</rdfs:comment>
        <rdfs:label xml:lang="en">has a super-property</rdfs:label>
        <rdfs:label xml:lang="fr">avoir une super-propriété</rdfs:label>
</rdf:Property>

<!-- 2.6.CreatePropertyDomainLink [param1:hasPropertyDomainCreate]
[param2:hasConceptDomainCreate] -->
<rdfs:Class rdf:ID="CreatePropertyDomainLink">
        <rdfs:subClassOf rdf:resource="#E_PropertyChange"/>
        <rdfs:comment xml:lang="en">create a domain link between a concept and a
property</rdfs:comment>
        <rdfs:comment xml:lang="fr">créer un lien domaine entre un concept et une
propriété</rdfs:comment>
        <rdfs:label xml:lang="en">create a property domain link</rdfs:label>
        <rdfs:label xml:lang="fr">créer un lien domaine de propriété</rdfs:label>
</rdfs:Class>

<rdf:Property rdf:ID="hasPropertyDomainCreate">
        <rdfs:domain rdf:resource="#CreatePropertyDomainLink"/>
```

```
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
        <rdfs:comment xml:lang="en">the property to be linked with concept
domain</rdfs:comment>
        <rdfs:comment xml:lang="fr">la propriété à lier au concept domaine</rdfs:comment>
        <rdfs:label xml:lang="en">has a property</rdfs:label>
        <rdfs:label xml:lang="fr">avoir une propriété</rdfs:label>
</rdf:Property>

<rdf:Property rdf:ID="hasConceptDomainCreate">
        <rdfs:domain rdf:resource="#CreatePropertyDomainLink"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Class"/>
        <rdfs:comment xml:lang="en">the concept to become the domain of
property</rdfs:comment>
        <rdfs:comment xml:lang="fr">le concept qui devient le domaine de la
propriété</rdfs:comment>
        <rdfs:label xml:lang="en">has a concept</rdfs:label>
        <rdfs:label xml:lang="fr">avoir un concept</rdfs:label>
</rdf:Property>

<!-- 2.7.RemovePropertyDomainLink [param1:hasPropertyDomainRemove]
[param2:hasConceptDomainRemove] -->
<rdfs:Class rdf:ID="RemovePropertyDomainLink">
        <rdfs:subClassOf rdf:resource="#E_PropertyChange"/>
        <rdfs:comment xml:lang="en">remove a domain link between a concept and a
property</rdfs:comment>
        <rdfs:comment xml:lang="fr">enlever un lien domaine entre un concept et une
propriété</rdfs:comment>
        <rdfs:label xml:lang="en">remove a property domain link</rdfs:label>
        <rdfs:label xml:lang="fr">enlever un lien domaine de propriété</rdfs:label>
</rdfs:Class>

<rdf:Property rdf:ID="hasPropertyDomainRemove">
        <rdfs:domain rdf:resource="#RemovePropertyDomainLink"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
        <rdfs:comment xml:lang="en">the property to be removed with concept
domain</rdfs:comment>
        <rdfs:comment xml:lang="fr">la propriété dont le concept ne sera plus concept
domaine</rdfs:comment>
        <rdfs:label xml:lang="en">has a property</rdfs:label>
        <rdfs:label xml:lang="fr">avoir une propriété</rdfs:label>
</rdf:Property>

<rdf:Property rdf:ID="hasConceptDomainRemove">
        <rdfs:domain rdf:resource="#RemovePropertyDomainLink"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Class"/>
        <rdfs:comment xml:lang="en">the concept to be removed from the domain of
property</rdfs:comment>
        <rdfs:comment xml:lang="fr">le concept qui sera enlevé du domaine de la
propriété</rdfs:comment>
        <rdfs:label xml:lang="en">has a concept</rdfs:label>
        <rdfs:label xml:lang="fr">avoir un concept</rdfs:label>
</rdf:Property>

<!-- 2.8.CreatePropertyRangeLink [param1:hasPropertyRangeCreate]
[param2:hasConceptRangeCreate] -->
<rdfs:Class rdf:ID="CreatePropertyRangeLink">
        <rdfs:subClassOf rdf:resource="#E_PropertyChange"/>
        <rdfs:comment xml:lang="en">create a range link between a concept and a
property</rdfs:comment>
        <rdfs:comment xml:lang="fr">créer un lien co-domaine entre un concept et une
propriété</rdfs:comment>
        <rdfs:label xml:lang="en">create a property range link</rdfs:label>
        <rdfs:label xml:lang="fr">créer un lien co-domaine de propriété</rdfs:label>
</rdfs:Class>

<rdf:Property rdf:ID="hasPropertyRangeCreate">
        <rdfs:domain rdf:resource="#CreatePropertyRangeLink"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
        <rdfs:comment xml:lang="en">the property to be linked with concept
range</rdfs:comment>
        <rdfs:comment xml:lang="fr">la propriété à lier au concept
codomaine</rdfs:comment>
        <rdfs:label xml:lang="en">has a property</rdfs:label>
        <rdfs:label xml:lang="fr">avoir une propriété</rdfs:label>
</rdf:Property>

<rdf:Property rdf:ID="hasConceptRangeCreate">
        <rdfs:domain rdf:resource="#CreatePropertyRangeLink"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Class"/>
        <rdfs:comment xml:lang="en">the concept to become the range of
property</rdfs:comment>
        <rdfs:comment xml:lang="fr">le concept qui devient le co-domaine de la
propriété</rdfs:comment>
        <rdfs:label xml:lang="en">has a concept</rdfs:label>
        <rdfs:label xml:lang="fr">avoir un concept</rdfs:label>
```

```
</rdf:Property>

<!-- 2.9.RemovePropertyRangeLink [param1:hasPropertyRangeRemove]
[param2:hasConceptRangeRemove] -->
<rdfs:Class rdf:ID="RemovePropertyRangeLink">
        <rdfs:subClassOf rdf:resource="#E_PropertyChange"/>
        <rdfs:comment xml:lang="en">remove a range link between a concept and a
property</rdfs:comment>
        <rdfs:comment xml:lang="fr">enlever un lien co-domaine entre un concept et une
propriété</rdfs:comment>
        <rdfs:label xml:lang="en">remove a property range link</rdfs:label>
        <rdfs:label xml:lang="fr">enlever un lien co-domaine de propriété</rdfs:label>
</rdfs:Class>

<rdf:Property rdf:ID="hasPropertyRangeRemove">
        <rdfs:domain rdf:resource="#RemovePropertyRangeLink"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
        <rdfs:comment xml:lang="en">the property to be removed with concept
range</rdfs:comment>
        <rdfs:comment xml:lang="fr">la propriété dont le concept n'appartiendra plus au
co-domaine</rdfs:comment>
        <rdfs:label xml:lang="en">has a property</rdfs:label>
        <rdfs:label xml:lang="fr">avoir une propriété</rdfs:label>
</rdf:Property>

<rdf:Property rdf:ID="hasConceptRangeRemove">
        <rdfs:domain rdf:resource="#RemovePropertyRangeLink"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Class"/>
        <rdfs:comment xml:lang="en">the concept to be removed from the range of
property</rdfs:comment>
        <rdfs:comment xml:lang="fr">le concept qui sera enlevé du co-domaine de la
propriété</rdfs:comment>
        <rdfs:label xml:lang="en">has a concept</rdfs:label>
        <rdfs:label xml:lang="fr">avoir un concept</rdfs:label>
</rdf:Property>


<!-- 4. Description for Elementary CommentChange branch -->
<rdfs:Class rdf:ID="E_CommentChange">
        <rdfs:subClassOf rdf:resource="#ElementaryChange"/>
        <rdfs:comment xml:lang="en">An elementary ontology change for
comment</rdfs:comment>
        <rdfs:comment xml:lang="fr">Un changement ontologique elementaire pour le
commentaire</rdfs:comment>
        <rdfs:label xml:lang="en">an elementary change of comment</rdfs:label>
        <rdfs:label xml:lang="fr">un changement élémentaire sur le
commentaire</rdfs:label>
</rdfs:Class>

<!-- Composite changes are classified by Concept, Property changes -->
<!-- 5. Description for Composite ConceptChange branch -->
<rdfs:Class rdf:ID="C_ConceptChange">
        <rdfs:subClassOf rdf:resource="#CompositeChange"/>
        <rdfs:comment xml:lang="en">A composite ontology change for
concept</rdfs:comment>
        <rdfs:comment xml:lang="fr">Un changement ontologique composite pour le
concept</rdfs:comment>
        <rdfs:label xml:lang="en">a composite change on concept</rdfs:label>
        <rdfs:label xml:lang="fr">un changement composite sur le concept</rdfs:label>
</rdfs:Class>

<!-- 5.1.CreateCommonConcept [param1:hasCommonConcept1] [param2:hasCommonConcept2]
[param3:hasCommonConceptNew] -->
<rdfs:Class rdf:ID="CreateCommonConcept">
        <rdfs:subClassOf rdf:resource="#C_ConceptChange"/>
        <rdfs:comment xml:lang="en">create a father concept commun of 2 existing
concepts</rdfs:comment>
        <rdfs:comment xml:lang="fr">créer un concept père commun de 2 concepts
existants</rdfs:comment>
        <rdfs:label xml:lang="en">create a common concept</rdfs:label>
        <rdfs:label xml:lang="fr">créer un concept commun</rdfs:label>
</rdfs:Class>

<rdf:Property rdf:ID="hasCommonConcept1">
        <rdfs:domain rdf:resource="#CreateCommonConcept"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Class"/>
        <rdfs:comment xml:lang="en">the 1st concept</rdfs:comment>
        <rdfs:comment xml:lang="fr">le 1er concept</rdfs:comment>
        <rdfs:label xml:lang="en">has the first concept</rdfs:label>
        <rdfs:label xml:lang="fr">avoir le premier concept</rdfs:label>
</rdf:Property>

<rdf:Property rdf:ID="hasCommonConcept2">
        <rdfs:domain rdf:resource="#CreateCommonConcept"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Class"/>
```

```
        <rdfs:comment xml:lang="en">the 2nd concept</rdfs:comment>
        <rdfs:comment xml:lang="fr">le 2eme concept</rdfs:comment>
        <rdfs:label xml:lang="en">has the second concept</rdfs:label>
        <rdfs:label xml:lang="fr">avoir la deuxième concept</rdfs:label>
</rdf:Property>

<rdf:Property rdf:ID="hasCommonConceptNew">
        <rdfs:domain rdf:resource="#CreateCommonConcept"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Class"/>
        <rdfs:comment xml:lang="en">the father concept commun created</rdfs:comment>
        <rdfs:comment xml:lang="fr">le concept pere commun cré</rdfs:comment>
        <rdfs:label xml:lang="en">has a common concept</rdfs:label>
        <rdfs:label xml:lang="fr">avoir un concept commun</rdfs:label>
</rdf:Property>

<!-- 5.2.MergeConcept [param1:hasMergeConcept1] [param2:hasMergeConcept2]
[param3:hasMergeConceptNew] -->
<rdfs:Class rdf:ID="MergeConcept">
        <rdfs:subClassOf rdf:resource="#C_ConceptChange"/>
        <rdfs:comment xml:lang="en">merge 2 existing concepts to a new one
</rdfs:comment>
        <rdfs:comment xml:lang="fr">fusionner 2 concepts existants dans un nouveau
concept</rdfs:comment>
        <rdfs:label xml:lang="en">merge two concepts</rdfs:label>
        <rdfs:label xml:lang="fr">fusionner deux concepts</rdfs:label>
</rdfs:Class>

<rdf:Property rdf:ID="hasMergeConcept1">
        <rdfs:domain rdf:resource="#MergeConcept"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Class"/>
        <rdfs:comment xml:lang="en">the 1st concept to be merged</rdfs:comment>
        <rdfs:comment xml:lang="fr">le 1er concept qui sera fusionné</rdfs:comment>
        <rdfs:label xml:lang="en">has the first concept to merge</rdfs:label>
        <rdfs:label xml:lang="fr">avoir le premier concept à fusinner</rdfs:label>
</rdf:Property>

<rdf:Property rdf:ID="hasMergeConcept2">
        <rdfs:domain rdf:resource="#MergeConcept"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Class"/>
        <rdfs:comment xml:lang="en">the 2nd concept to be merged</rdfs:comment>
        <rdfs:comment xml:lang="fr">le 2eme concept qui sera fusionné</rdfs:comment>
        <rdfs:label xml:lang="en">has the second concept to merge</rdfs:label>
        <rdfs:label xml:lang="fr">avoir la deuxième concept à fusionner</rdfs:label>
</rdf:Property>

<rdf:Property rdf:ID="hasMergeConceptNew">
        <rdfs:domain rdf:resource="#MergeConcept"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Class"/>
        <rdfs:comment xml:lang="en">the merged concept</rdfs:comment>
        <rdfs:comment xml:lang="fr">le concept fusionne</rdfs:comment>
        <rdfs:label xml:lang="en">has a merged concept</rdfs:label>
        <rdfs:label xml:lang="fr">avoir un concept fusionné</rdfs:label>
</rdf:Property>

<!-- 5.3.SplitConcept [param1:hasConceptSplit] [param2:hasSplittedConcept1]
[param3:hasSplittedConcept2] -->
<rdfs:Class rdf:ID="SplitConcept">
        <rdfs:subClassOf rdf:resource="#C_ConceptChange"/>
        <rdfs:comment xml:lang="en">split one existing concept to 2 new concepts
</rdfs:comment>
        <rdfs:comment xml:lang="fr">diviser un concept existant dans 2 nouveaux
concepts</rdfs:comment>
        <rdfs:label xml:lang="en">split a concept</rdfs:label>
        <rdfs:label xml:lang="fr">diviser un concept</rdfs:label>
</rdfs:Class>

<rdf:Property rdf:ID="hasConceptSplit">
        <rdfs:domain rdf:resource="#SplitConcept"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Class"/>
        <rdfs:comment xml:lang="en">the concept to be splitted</rdfs:comment>
        <rdfs:comment xml:lang="fr">le concept qui sera divisé</rdfs:comment>
        <rdfs:label xml:lang="en">has a concept to split</rdfs:label>
        <rdfs:label xml:lang="fr">avoir un concept à diviser</rdfs:label>
</rdf:Property>

<rdf:Property rdf:ID="hasSplittedConcept1">
        <rdfs:domain rdf:resource="#SplitConcept"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Class"/>
        <rdfs:comment xml:lang="en">the 1st concept splitted</rdfs:comment>
        <rdfs:comment xml:lang="fr">le 1er concept divisé</rdfs:comment>
        <rdfs:label xml:lang="en">has the first concept splitted</rdfs:label>
        <rdfs:label xml:lang="fr">avoir le premier concept divisé</rdfs:label>
</rdf:Property>

<rdf:Property rdf:ID="hasSplittedConcept2">
```

```
        <rdfs:domain rdf:resource="#SplitConcept"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Class"/>
        <rdfs:comment xml:lang="en">the 2nd concept splitted</rdfs:comment>
        <rdfs:comment xml:lang="fr">le 2eme concept divisé</rdfs:comment>
        <rdfs:label xml:lang="en">has the second concept splitted</rdfs:label>
        <rdfs:label xml:lang="fr">avoir le deuxième concept divisé</rdfs:label>
</rdf:Property>

<!-- 5.4.MoveConcept [param1:hasMoveConcept] [param2:hasMoveConceptReference] -->
<rdfs:Class rdf:ID="MoveConcept">
        <rdfs:subClassOf rdf:resource="#C_ConceptChange"/>
        <rdfs:comment xml:lang="en">move one existing concept to another concept (as
child of this concept) </rdfs:comment>
        <rdfs:comment xml:lang="fr">déplacer un concept existant vers un autre concept
(comme le fils de ce concept)</rdfs:comment>
        <rdfs:label xml:lang="en">move a concept</rdfs:label>
        <rdfs:label xml:lang="fr">déplacer un concept</rdfs:label>
</rdfs:Class>

<rdf:Property rdf:ID="hasMoveConcept">
        <rdfs:domain rdf:resource="#MoveConcept"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Class"/>
        <rdfs:comment xml:lang="en">the concept to be moved</rdfs:comment>
        <rdfs:comment xml:lang="fr">le concept qui sera déplacé</rdfs:comment>
        <rdfs:label xml:lang="en">has a concept to be moved</rdfs:label>
        <rdfs:label xml:lang="fr">avoir un concept à déplacer</rdfs:label>
</rdf:Property>

<rdf:Property rdf:ID="hasMoveConceptReference">
        <rdfs:domain rdf:resource="#MoveConcept"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Class"/>
        <rdfs:comment xml:lang="en">the moved concept will be chid of this referent
concept</rdfs:comment>
        <rdfs:comment xml:lang="fr">le concept déplacé est le fils de ce concept referent
</rdfs:comment>
        <rdfs:label xml:lang="en">has a concept reference</rdfs:label>
        <rdfs:label xml:lang="fr">avoir un concept référent</rdfs:label>
</rdf:Property>

<!-- 5.5.InsertConceptGeneralisation [param1:hasNewConceptGeneralisation]
[param2:hasConceptGenReference] -->
<rdfs:Class rdf:ID="InsertConceptGeneralisation">
        <rdfs:subClassOf rdf:resource="#C_ConceptChange"/>
        <rdfs:comment xml:lang="en">insert a new concept newC between a given concept c
and all its parents</rdfs:comment>
        <rdfs:comment xml:lang="fr">insérer un nouveau concept newC entre un concept c et
tous ses concepts pères</rdfs:comment>
        <rdfs:label xml:lang="en">insert a generalised concept</rdfs:label>
        <rdfs:label xml:lang="fr">insérer un concept généralisé</rdfs:label>
</rdfs:Class>

<rdf:Property rdf:ID="hasNewConceptGeneralisation">
        <rdfs:domain rdf:resource="#InsertConceptGeneralisation"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Class"/>
        <rdfs:comment xml:lang="en">the generalised concept to be inserted</rdfs:comment>
        <rdfs:comment xml:lang="fr">le concept généralisé qui sera inséré</rdfs:comment>
        <rdfs:label xml:lang="en">has a generalised concept</rdfs:label>
        <rdfs:label xml:lang="fr">avoir un concept généralisé</rdfs:label>
</rdf:Property>

<rdf:Property rdf:ID="hasConceptGenReference">
        <rdfs:domain rdf:resource="#InsertConceptGeneralisation"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Class"/>
        <rdfs:comment xml:lang="en">the inserted concept generalisation will be father of
this referent concept</rdfs:comment>
        <rdfs:comment xml:lang="fr">le concept généralisé inséré est le père de ce
concept referent </rdfs:comment>
        <rdfs:label xml:lang="en">has a reference concept</rdfs:label>
        <rdfs:label xml:lang="fr">avoir un concept référent</rdfs:label>
</rdf:Property>

<!-- 5.6.InsertConceptSpecialisation [param1:hasNewConceptSpecialisation]
[param2:hasConceptSpeReference] -->
<rdfs:Class rdf:ID="InsertConceptSpecialisation">
        <rdfs:subClassOf rdf:resource="#C_ConceptChange"/>
        <rdfs:comment xml:lang="en">insert a new concept newC between a given concept c
and all its children</rdfs:comment>
        <rdfs:comment xml:lang="fr">insérer un nouveau concept newC entre un concept c et
tous ses concepts fils</rdfs:comment>
        <rdfs:label xml:lang="en">insert a specialised concept</rdfs:label>
        <rdfs:label xml:lang="fr">insérer un concept spécialisé</rdfs:label>
</rdfs:Class>

<rdf:Property rdf:ID="hasNewConceptSpecialisation">
        <rdfs:domain rdf:resource="#InsertConceptSpecialisation"/>
```

```
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Class"/>
        <rdfs:comment xml:lang="en">the concept specialisation to be
inserted</rdfs:comment>
        <rdfs:comment xml:lang="fr">le concept specialisé qui sera inséré</rdfs:comment>
        <rdfs:label xml:lang="en">has a specialised concept</rdfs:label>
        <rdfs:label xml:lang="fr">avoir un concept spécialisé</rdfs:label>
</rdf:Property>

<rdf:Property rdf:ID="hasConceptSpeReference">
        <rdfs:domain rdf:resource="#InsertConceptSpecialisation"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Class"/>
        <rdfs:comment xml:lang="en">the inserted concept generalisation will be child of
this referent concept</rdfs:comment>
        <rdfs:comment xml:lang="fr">le concept spécialisé inséré est le fils de ce
concept référent </rdfs:comment>
        <rdfs:label xml:lang="en">has a reference concept</rdfs:label>
        <rdfs:label xml:lang="fr">avoir un concept référent</rdfs:label>
</rdf:Property>


<!-- 6. Description for Composite PropertyChange branch -->
<rdfs:Class rdf:ID="C_PropertyChange">
        <rdfs:subClassOf rdf:resource="#CompositeChange"/>
        <rdfs:comment xml:lang="en">A composite ontology change for
property</rdfs:comment>
        <rdfs:comment xml:lang="fr">Un changement ontologique composite pour la
propriété</rdfs:comment>
        <rdfs:label xml:lang="en">a composite change on property</rdfs:label>
        <rdfs:label xml:lang="fr">un changement composite sur la propriété</rdfs:label>
</rdfs:Class>

<!-- 6.1.CreateCommonProperty [param1:hasCommonProperty1] [param2:hasCommonProperty2]
[param3:hasCommonPropertyNew] -->
<rdfs:Class rdf:ID="CreateCommonProperty">
        <rdfs:subClassOf rdf:resource="#C_PropertyChange"/>
        <rdfs:comment xml:lang="en">create a common mother property of 2 existing
properties</rdfs:comment>
        <rdfs:comment xml:lang="fr">créer une propriété mère commune de 2 propriétés
existantes</rdfs:comment>
        <rdfs:label xml:lang="en">create a common property</rdfs:label>
        <rdfs:label xml:lang="fr">créer une propriété commune</rdfs:label>
</rdfs:Class>

<rdf:Property rdf:ID="hasCommonProperty1">
        <rdfs:domain rdf:resource="#CreateCommonProperty"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
        <rdfs:comment xml:lang="en">the 1st property</rdfs:comment>
        <rdfs:comment xml:lang="fr">la 1ère propriété</rdfs:comment>
        <rdfs:label xml:lang="en">has the first property</rdfs:label>
        <rdfs:label xml:lang="fr">avoir pour première propriété</rdfs:label>
</rdf:Property>

<rdf:Property rdf:ID="hasCommonProperty2">
        <rdfs:domain rdf:resource="#CreateCommonProperty"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
        <rdfs:comment xml:lang="en">the 2nd property</rdfs:comment>
        <rdfs:comment xml:lang="fr">la 2ème propriété</rdfs:comment>
        <rdfs:label xml:lang="en">has the second property</rdfs:label>
        <rdfs:label xml:lang="fr">avoir pour deuxième propriété</rdfs:label>
</rdf:Property>

<rdf:Property rdf:ID="hasCommonPropertyNew">
        <rdfs:domain rdf:resource="#CreateCommonProperty"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
        <rdfs:comment xml:lang="en">the common mother property created</rdfs:comment>
        <rdfs:comment xml:lang="fr">la propriété mère commune créée</rdfs:comment>
        <rdfs:label xml:lang="en">has a common property</rdfs:label>
        <rdfs:label xml:lang="fr">avoir une propriété commune</rdfs:label>
</rdf:Property>

<!-- 6.2.MergeProperty [param1:hasMergeProperty1] [param2:hasMergeProperty2]
[param3:hasMergePropertyNew] -->
<rdfs:Class rdf:ID="MergeProperty">
        <rdfs:subClassOf rdf:resource="#C_PropertyChange"/>
        <rdfs:comment xml:lang="en">merge 2 existing properties to a new one
</rdfs:comment>
        <rdfs:comment xml:lang="fr">fusionner 2 propriétés existantes dans une nouvelle
propriété</rdfs:comment>
        <rdfs:label xml:lang="en">merge two properties</rdfs:label>
        <rdfs:label xml:lang="fr">fusionner deux propriétés</rdfs:label>
</rdfs:Class>

<rdf:Property rdf:ID="hasMergeProperty1">
        <rdfs:domain rdf:resource="#MergeProperty"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
```

```
        <rdfs:comment xml:lang="en">the 1st property to be merged</rdfs:comment>
        <rdfs:comment xml:lang="fr">la 1ère propriété qui sera fusionnée</rdfs:comment>
        <rdfs:label xml:lang="en">has the first property</rdfs:label>
        <rdfs:label xml:lang="fr">avoir pour première propriété</rdfs:label>
</rdf:Property>

<rdf:Property rdf:ID="hasMergeProperty2">
        <rdfs:domain rdf:resource="#MergeProperty"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
        <rdfs:comment xml:lang="en">the 2nd property to be merged</rdfs:comment>
        <rdfs:comment xml:lang="fr">la 2eme propriété qui sera fusionnée</rdfs:comment>
        <rdfs:label xml:lang="en">has the second property</rdfs:label>
        <rdfs:label xml:lang="fr">avoir pour deuxième propriété</rdfs:label>
</rdf:Property>

<rdf:Property rdf:ID="hasMergePropertyNew">
        <rdfs:domain rdf:resource="#MergeProperty"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
        <rdfs:comment xml:lang="en">the merged property</rdfs:comment>
        <rdfs:comment xml:lang="fr">la propiété fusionnée</rdfs:comment>
        <rdfs:label xml:lang="en">has a merged property</rdfs:label>
        <rdfs:label xml:lang="fr">avoir une propriété fusionnée</rdfs:label>
</rdf:Property>

<!-- 6.3.SplitProperty [param1:hasPropertySplit] [param2:hasSplittedProperty1]
[param3:hasSplittedProperty2] -->
<rdfs:Class rdf:ID="SplitProperty">
        <rdfs:subClassOf rdf:resource="#C_PropertyChange"/>
        <rdfs:comment xml:lang="en">split one existing property to 2 new properties
</rdfs:comment>
        <rdfs:comment xml:lang="fr">diviser une propriété existante dans 2 nouvelles
propriétés</rdfs:comment>
        <rdfs:label xml:lang="en">split a property</rdfs:label>
        <rdfs:label xml:lang="fr">diviser une propriété</rdfs:label>
</rdfs:Class>

<rdf:Property rdf:ID="hasPropertySplit">
        <rdfs:domain rdf:resource="#SplitProperty"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
        <rdfs:comment xml:lang="en">the property to be splitted</rdfs:comment>
        <rdfs:comment xml:lang="fr">la propriété qui sera divisée</rdfs:comment>
        <rdfs:label xml:lang="en">has a property to split</rdfs:label>
        <rdfs:label xml:lang="fr">avoir une propriété à diviser</rdfs:label>
</rdf:Property>

<rdf:Property rdf:ID="hasSplittedProperty1">
        <rdfs:domain rdf:resource="#SplitProperty"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
        <rdfs:comment xml:lang="en">the 1st property splitted</rdfs:comment>
        <rdfs:comment xml:lang="fr">la 1ère propriété divisée</rdfs:comment>
        <rdfs:label xml:lang="en">has the first property splitted</rdfs:label>
        <rdfs:label xml:lang="fr">avoir pour première propriété divisée</rdfs:label>
</rdf:Property>

<rdf:Property rdf:ID="hasSplittedProperty2">
        <rdfs:domain rdf:resource="#SplitProperty"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
        <rdfs:comment xml:lang="en">the 2nd property splitted</rdfs:comment>
        <rdfs:comment xml:lang="fr">la 2eme propriété divisée</rdfs:comment>
        <rdfs:label xml:lang="en">has the second property splitted</rdfs:label>
        <rdfs:label xml:lang="fr">avoir pour deuxième propriété divisée</rdfs:label>
</rdf:Property>

<!-- 6.4.MoveProperty [param1:hasMoveProperty] [param2:hasMovePropertyReference] -->
<rdfs:Class rdf:ID="MoveProperty">
        <rdfs:subClassOf rdf:resource="#C_PropertyChange"/>
        <rdfs:comment xml:lang="en">move one existing property to another property (as
child of this property) </rdfs:comment>
        <rdfs:comment xml:lang="fr">déplacer une propriété existante vers une autre
propriété (comme la fille de cette propriété)</rdfs:comment>
        <rdfs:label xml:lang="en">move a property</rdfs:label>
        <rdfs:label xml:lang="fr">déplacer une propriété</rdfs:label>
</rdfs:Class>

<rdf:Property rdf:ID="hasMoveProperty">
        <rdfs:domain rdf:resource="#MoveProperty"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
        <rdfs:comment xml:lang="en">the property to be moved</rdfs:comment>
        <rdfs:comment xml:lang="fr">la propriété qui sera déplacée</rdfs:comment>
        <rdfs:label xml:lang="en">has a property to be moved</rdfs:label>
        <rdfs:label xml:lang="fr">avoir une propriété à déplacer</rdfs:label>
</rdf:Property>

<rdf:Property rdf:ID="hasMovePropertyReference">
        <rdfs:domain rdf:resource="#MoveProperty"/>
```

```
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
        <rdfs:comment xml:lang="en">the moved property will be chid of this referent
property</rdfs:comment>
        <rdfs:comment xml:lang="fr">la propriété déplacée est la fille de cette propriété
referente </rdfs:comment>
        <rdfs:label xml:lang="en">has a reference property</rdfs:label>
        <rdfs:label xml:lang="fr">avoir une propriété référente</rdfs:label>
</rdf:Property>

<!-- 6.5.InsertPropertyGeneralisation [param1:hasNewPropertyGeneralisation]
[param2:hasPropertyGenReference] -->
<rdfs:Class rdf:ID="InsertPropertyGeneralisation">
        <rdfs:subClassOf rdf:resource="#C_PropertyChange"/>
        <rdfs:comment xml:lang="en">insert a new property newP between a given property p
and all its parents</rdfs:comment>
        <rdfs:comment xml:lang="fr">insérer une nouvelle propriété newP entre une
propriété p et tous ses propriétés mères</rdfs:comment>
        <rdfs:label xml:lang="en">insert a generalised property</rdfs:label>
        <rdfs:label xml:lang="fr">insérer une propriété généralisée</rdfs:label>
</rdfs:Class>

<rdf:Property rdf:ID="hasNewPropertyGeneralisation">
        <rdfs:domain rdf:resource="#InsertPropertyGeneralisation"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
        <rdfs:comment xml:lang="en">the property generalisation to be
inserted</rdfs:comment>
        <rdfs:comment xml:lang="fr">la propriété generalisation qui sera
insérée</rdfs:comment>
        <rdfs:label xml:lang="en">has a generalised property</rdfs:label>
        <rdfs:label xml:lang="fr">avoir une propriété généralisée</rdfs:label>
</rdf:Property>

<rdf:Property rdf:ID="hasPropertyGenReference">
        <rdfs:domain rdf:resource="#InsertPropertyGeneralisation"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
        <rdfs:comment xml:lang="en">the inserted property generalisation will be mother
of this referent property</rdfs:comment>
        <rdfs:comment xml:lang="fr">la propriété généralisation insérée est la mère de
cette propriété referente </rdfs:comment>
        <rdfs:label xml:lang="en">has a reference property</rdfs:label>
        <rdfs:label xml:lang="fr">avoir une propriété référente</rdfs:label>
</rdf:Property>

<!-- 6.6.InsertPropertySpecialisation [param1:hasNewPropertySpecialisation]
[param2:hasPropertySpeReference] -->
<rdfs:Class rdf:ID="InsertPropertySpecialisation">
        <rdfs:subClassOf rdf:resource="#C_PropertyChange"/>
        <rdfs:comment xml:lang="en">insert a new property newP between a given property p
and all its children</rdfs:comment>
        <rdfs:comment xml:lang="fr">insérer une nouvelle propriété newP entre une
propiété p et toutes ses propriétés filles</rdfs:comment>
        <rdfs:label xml:lang="en">insert a specialised property</rdfs:label>
        <rdfs:label xml:lang="fr">insérer une propriété spécialisée</rdfs:label>
</rdfs:Class>

<rdf:Property rdf:ID="hasNewPropertySpecialisation">
        <rdfs:domain rdf:resource="#InsertPropertySpecialisation"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
        <rdfs:comment xml:lang="en">the property specialisation to be
inserted</rdfs:comment>
        <rdfs:comment xml:lang="fr">la propriété spécialisée qui sera
insérée</rdfs:comment>
        <rdfs:label xml:lang="en">has a specialised property</rdfs:label>
        <rdfs:label xml:lang="fr">avoir une propriété spécialisée</rdfs:label>
</rdf:Property>

<rdf:Property rdf:ID="hasPropertySpeReference">
        <rdfs:domain rdf:resource="#InsertPropertySpecialisation"/>
        <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
        <rdfs:comment xml:lang="en">the inserted property generalisation will be child of
this referent property</rdfs:comment>
        <rdfs:comment xml:lang="fr">la propriété spécialisée insérée est la fille de
cette propriété referente </rdfs:comment>
        <rdfs:label xml:lang="en">has a reference property</rdfs:label>
        <rdfs:label xml:lang="fr">avoir une propriété référente</rdfs:label>
</rdf:Property>

</rdf:RDF>
```

Annexe C – Exemple d'une trace d'évolution

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xml:base="http://www.inria.fr/acacia/coswem#"
  xmlns:csw="http://www.inria.fr/acacia/coswem#"
  xmlns:ev="http://www.inria.fr/acacia/evolution#"
  xmlns:geo="http://rdf.insee.fr/geo#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

<csw:TraceOnto rdf:about="http://www.inria.fr/acacia/coswem#TraceO1">
  <csw:hasVersionBefore>COG-v1</csw:hasVersionBefore>
    <csw:hasVersionAfter>COG-v2</csw:hasVersionAfter>
        <csw:hasChange>
        <rdf:Bag>
          <rdf:li>
           <csw:DeleteConcept rdf:about="http://www.inria.fr/acacia/evolution#id-
deleteconcept01">
              <csw:hasDeleteConcept rdf:resource="http://rdf.insee.fr/geo#Ville_Commune"/>
               <csw:hasAnnotInconsistency>yes</csw:hasAnnotInconsistency>
               <csw:hasAdditionalChange>
                 <csw:CreateHierarchyConceptLink
rdf:about="http://www.inria.fr/acacia/coswem#id-createhierarchyconceptlink01">
                  <csw:hasSubConceptCreate rdf:resource="http://rdf.insee.fr/geo#Ville"/>
                                                    <csw:hasSuperConceptCreate
rdf:resource="http://rdf.insee.fr/geo#Territoire_FR"/>
                   <csw:hasAnnotInconsistency>no</csw:hasAnnotInconsistency>
                 </csw:CreateHierarchyConceptLink>
               </csw:hasAdditionalChange>
            </csw:DeleteConcept>
        </rdf:li>
        <rdf:li>
        <csw:RenameConcept rdf:about="http://www.inria.fr/acacia/evolution#id-renameconcept01">
            <csw:hasOldConcept rdf:resource="http://rdf.insee.fr/geo#Canton"/>
            <csw:hasNewConceptName>Canton-Administrative</csw:hasNewConceptName>
             <csw:hasAnnotInconsistency>yes</csw:hasAnnotInconsistency>
          </csw:RenameConcept>
        </rdf:li>
        <rdf:li>
           <csw:InsertConcept rdf:about="http://www.inria.fr/acacia/evolution#id-
insertconcept01">
            <csw:hasInsertConcept>Capital</csw:hasInsertConcept>
            <csw:hasInsertConceptReference
rdf:resource="http://rdf.insee.fr/geo#Territoire_FR"/>
            <csw:hasAnnotInconsistency>no</csw:hasAnnotInconsistency>
          </csw:InsertConcept>
        </rdf:li>
        <rdf:li>
          <csw:CreateConceptDomainLink rdf:about="http://www.inria.fr/acacia/coswem#id-
createconceptdomainlink01">
            <csw:hasConceptDomainCreate rdf:resource="http://rdf.insee.fr/geo#Capital"/>
            <csw:hasPropertyDomainCreate
rdf:resource="http://rdf.insee.fr/geo#role_administratif"/>
             <csw:hasAnnotInconsistency>no</csw:hasAnnotInconsistency>
          </csw:CreateConceptDomainLink>
        </rdf:li>
        <rdf:li>
           <csw:RemoveConceptDomainLink rdf:about="http://www.inria.fr/acacia/coswem#id-
createconceptdomainlink02">
            <csw:hasConceptDomainRemove rdf:resource="http://rdf.insee.fr/geo#Ville"/>
                <csw:hasPropertyDomainRemove
rdf:resource="http://rdf.insee.fr/geo#role_administratif_ville"/>
                <csw:hasAnnotInconsistency>yes</csw:hasAnnotInconsistency>
           </csw:RemoveConceptDomainLink>
        </rdf:li>
   </rdf:Bag>
</csw:hasChange>
</csw:TraceOnto>
</rdf:RDF>
```

Résumé

Le Web Sémantique d'Entreprise (WSE) est une approche particulière de la Gestion des Connaissances d'une entreprise pour la prochaine génération du Web Sémantique dans laquelle on donne à une information un sens bien défini pour permettre aux ordinateurs et aux utilisateurs de travailler en coopération. Dans la réalité, les organisations vivent dans un environnement hétérogène, dynamique et en cours d'évolution qui mène souvent à des changements externes et internes requérant l'évolution de leur système de gestion des connaissances. Peu de recherches actuelles font face aux changements et fournissent des facilités de maintenance pour le système de gestion des connaissances. L'objectif de cette thèse est de contribuer à lever cette limitation.
Dans ce manuscrit, nous présentons une nouvelle approche de la gestion de l'évolution du WSE. Nous nous focalisons en particulier sur l'évolution de l'ontologie et de l'annotation sémantique qui sont deux composants importants du WSE. Nous nous intéressons à deux scénarios d'évolution de l'ontologie : (i) avec trace et (ii) sans trace de changements ontologiques effectués. Ces deux scénarios sont fréquents dans les situations réelles et ils peuvent entraîner des inconsistances au niveau des annotations sémantiques reposant sur cette ontologie modifiée. Pour chacun des contextes d'évolution, nous développons des approches équivalentes : une approche procédurale et une approche basée sur des règles en vue de gérer l'évolution des annotations sémantiques et, en particulier, de détecter et de corriger les annotations sémantiques inconsistantes.
Ces propositions ont été implémentées et validées dans le système CoSWEM qui facilite la gestion de l'évolution du WSE en permettant de réaliser certaines tâches d'une manière automatique ou semi-automatique telles que la comparaison d'ontologies différentes, la détection et la correction des inconsistances sur les annotations sémantiques, etc. Ce système a été expérimenté dans le cadre des projets PALETTE et E-WOK_HUB sur un ensemble de données réelles et évolutives provenant de ces projets.

Mots-clés : web sémantique, évolution, web sémantique d'entreprise, ontologie, annotation sémantique, règles, détection d'inconsistance, résolution d'inconsistance, stratégie de résolution, RDF(S).

Abstract

Corporate Semantic Web (CSW) is an approach of the Knowledge Management in an organisation for the next generation of the Semantic Web in which information is given well-defined meaning, better enabling computers and people to work in co-operation. Actually, the organizations live in a heterogeneous, dynamic and evolving environment which often leads to internal and external changes requiring the evolution of their knowledge management system. Few current researches face the change management problem and provide the maintenance facilities for the knowledge management system. The objective of this thesis is to contribute to reducing this limitation.
In this dissertation, we present a new approach for the evolution management of the CSW. We focus particularly on the ontology evolution and semantic annotation evolution which are two important components of the CSW. We are interested in two main scenarios of ontology evolution : (i) with trace and (ii) without trace of ontology changes which are carried out during its evolution. These two scenarios are often encountered in reality and they lead to inconsistencies of the annotations semantics using this modified ontology. Corresponding to each context of evolution, we have developed equivalent approaches: a procedural approach and a rule-based approach in order to manage semantic annotations evolution and particularly to detect inconsistent annotations and to guide the process of solving these inconsistencies.
These propositions were implemented and validated in the CoSWEM system which facilitates the evolution management of the CSW. It enables to carry out some tasks automatically or semi-automatically such as comparison of different ontologies, inconsistency detection and correction of the semantic annotations, etc. This system was also experimented within the framework of the industrial projects PALETTE and E-WOK_HUB with a set of real and evolving data from these projects.

Keywords : semantic web, evolution, corporate semantic web, ontology, semantic annotation, rules, inconsistency detection, inconsistency resolution, resolution strategy, RDF(S).

Thèse préparée à l'INRIA Sophia Antipolis dans le projet ACACIA/EDELWEISS