Construction d'un Web sémantique multi-points de vue

Lê Bach Thanh

année : 2006

N° attribué par la bibliothèque

THÈSE

présentée à

L'École des Mines de Paris à Sophia Antipolis

Sciences de l'ingénieur

pour obtenir le titre de

DOCTEUR EN SCIENCES

Spécialité

Informatique

soutenue par

BACH Thành Lê

le 23 octobre 2006

Titre

Construction d'un Web sémantique multi-points de vue

Thèse dirigée par **Rose Dieng-Kuntz** & **Franck Guarnieri**

*et préparée à l'***INRIA Sophia Antipolis**, *projet* **ACACIA**

Jury

Président	Henri Briand
Rapporteurs	Amedeo Napoli
	Nhan Le Thanh
Examinateurs	Rose Dieng-Kuntz
	Franck Guarnieri
	Sylvie Szulman

Résumé

Dans cette thèse, nous étudions les problèmes de l'hétérogénéité et du consensus dans un Web sémantique d'entreprise. Dans le *Web sémantique*, une extension du Web actuel, la *sémantique* des ressources est rendue *explicite* pour que les machines et les agents puissent les « *comprendre* » et les traiter *automatiquement*, afin de faciliter les tâches des utilisateurs finaux. Un *Web sémantique d'entreprise* est un tel web sémantique dédié à une entreprise, une organisation.

L'objectif de cette thèse est de permettre la construction et l'exploitation d'un tel Web sémantique, dans une organisation hétérogène comportant différentes sources de connaissances et différentes catégories d'utilisateurs, *sans éliminer* l'hétérogénéité mais en *faisant cohabiter* entre l'hétérogénéité et le consensus dans l'organisation tout entière.

Dans la première partie, nous approfondissons le problème de l'hétérogénéité des ontologies. L'*ontologie* est un des éléments fondamentaux dans le Web sémantique. Plusieurs ontologies différentes peuvent co-exister dans une organisation hétérogène. Pour faciliter l'échange des informations et des connaissances encodées dans différentes ontologies, nous étudions des algorithmes permettant d'*aligner* des ontologies déjà existantes. Les algorithmes proposés permettent de mettre en correspondance les ontologies représentées dans les langages RDF(S) et OWL recommandés par le W3C pour le Web sémantique. Ces algorithmes sont évalués grâce à des campagnes d'évaluation des outils d'alignement d'ontologies.

Dans la deuxième partie, nous nous intéressons au problème de la construction de *nouvelles* ontologies dans une organisation hétérogène mais en prenant en compte *différents points de vue*, différentes terminologies des personnes, des groupes voire des communautés diverses au sein de cette organisation. Une telle ontologie, appelée *ontologie multi-points de vue*, permet de faire cohabiter à la fois l'hétérogénéité et le consensus dans une organisation hétérogène. Nous proposons un modèle de représentation des connaissances multi-points de vue, appelé MVP, et un langage d'ontologie multi-points de vue, qui est une extension du langage d'ontologie OWL, appelé MVP-OWL, pour permettre de construire et d'exploiter des ontologies multi-points de vue dans un Web sémantique d'entreprise.

Mots-clés : Web sémantique, Web sémantique d'entreprise, ontologie, alignement d'ontologies, modèle multi-points de vue, ontologie multi-points de vue, RDF(S), OWL.

Abstract

In this thesis, we study problems of heterogeneity and consensus in a corporate semantic Web. In the *semantic Web*, an extension of the current Web, information is given *well-defined meaning* so that machines and agents can « *understand* » and process them *automatically*, in order to facilitate the end-user tasks. A *corporate semantic Web* is a semantic Web dedicated to an enterprise or an organization.

The objective of this thesis is to allow the construction and the exploitation of such semantic Web, in a heterogeneous organization comprising various sources of knowledge and various user categories, *without eliminating* heterogeneity but by making heterogeneity and consensus *cohabit* in the whole organization.

In the first part, we study thoroughly the ontology heterogeneity problem. *Ontology* is one of the most important fundamental elements in the semantic Web. Several different ontologies can coexist in a heterogeneous organization. To facilitate information exchange where information is encoded by various ontologies, we study algorithms allowing to *align* already existing ontologies. The proposed algorithms enable to align ontologies represented in different ontology languages recommended by W3C for the semantic Web: RDF(S) and OWL. These algorithms are evaluated through alignment tool evaluation campaigns.

In the second part, we are interested in the problem of building *new* ontologies in a heterogeneous organization by taking into account *different viewpoints*, different terminologies of various users, groups or even communities in the organization. Such ontology, called *multi-viewpoint ontology*, enables to have both heterogeneity and consensus co-exist in a heterogeneous organization. We propose a multi-viewpoint knowledge representation model, called MVP, and a multi-viewpoint ontology language, called MVP-OWL, which is an extension of the ontology language OWL, to allow the construction and the exploitation of the multi-viewpoint ontology in a corporate semantic Web.

Keywords: Semantic Web, corporate semantic Web, ontology, ontology alignment, model multi-viewpoint, multi-viewpoint ontology, RDF(S), OWL.

Remerciements

Je tiens à remercier ici M. Amedeo NAPOLI, *Directeur de recherche CNRS au LORIA*, ainsi que M. Nhan LE THANH, *Professeur à l'université de Nice-Sophia Antipolis*, qui ont accepté de rapporter ce travail et qui m'ont apporté des remarques enthousiastes et enrichissantes.

Je remercie également M. Henri BRIAND, *Professeur à l'université de Nantes*, qui m'a fait l'honneur de présider le jury de cette thèse, ainsi que Mme. Sylvie SZULMAN, *Maître de conférences à Université Paris-Nord*, pour sa participation en tant que membre du jury.

Je tiens à remercier vivement et profondément Mme. Rose DIENG-KUNTZ, *Directeur de recherche INRIA, Responsable scientifique du projet ACACIA*, pour m'avoir accueilli dans son équipe et pour la confiance et la liberté qu'elle a su m'accorder ainsi que pour ses précieux conseils et son support inestimable et inoubliable tout au long de cette thèse.

Je tiens également à remercier M. Franck GUARNIERI, *Directeur du Pôle Cindyniques de l'École des Mines de Paris à Sophia-Antipolis*, pour avoir accepté de co-encardrer mon travail et pour avoir toujours gardé confiance en moi.

J'adresse aussi tous mes remerciements à Olivier Corby pour ses remarques pertinentes, ses encouragements, à Alain Giboin pour nos discussions valables, et particulièrement à Fabien Gandon pour ses avis précieux. Tous les trois à leur manière ont su m'apporter l'aide nécessaire pour faire avancer ce travail et pour arriver jusqu'au bout de cette thèse.

Un remerciement à tous les membres présents du l'équipe ACACIA : Adil El Ghali, Amira Tifous, Aurelie Girardot, Hacene Cherfi, Laurent Alamarguy, Michel Buffa, Mohamed Bennis, Mohamed Semi Gaieb, Patricia Maleyran, Phuc-Hiep Luong, Priscille Durville, Sylvain Dehors, Tuan-Dung Cao, Virginie Bottollier, Yann-Vigile Hoareau ; et aux anciens membres : Carolina Medina, Cécile Guigard, Marek Ruzicka, Nicolas Gronnier, Olivier Savoie, Sophie Honnorat. Un remerciement tout particulier à Khaled Khelif qui a partagé le même bureau avec moi pendant ces longues années de thèse et qui m'a aidé beaucoup au niveau de la langue française ainsi qu'au niveau de la vie quotidienne.

Je remercie les membres du SEMIR, de la doc de l'INRIA Sophia-Antipolis, du service de restaurant, du poste de garde et les autres membres de l'INRIA

Sophia-Antipolis pour leur accueil, la qualité de leur travail, leur gentillesse et leurs sourires.

Un remerciement à tous les étudiants vietnamiens du groupe Nayno de la Côte d'Azur pour leurs encouragements, leurs aides. Particulièrement, un grand remerciement à Thịnh et sa famille, à Đức Duy, Hà Ú, à Anh Lê, Thu Hà et les autres qui ont partagé les événements mémorables avec moi pendant mon séjour en France.

Et finalement, un énorme remerciement à ma famille : mon père Thắng, ma mère Hồng, ma sœur Hương, mes frères Sơn, Khoa ainsi mon beau-père Chiến, ma belle-mère Oanh et mon beau-frère Tiến pour m'avoir compris et soutenu du mieux possible tout au long de ce travail. Je souhaite, *très spécialement*, remercier tout au fond de mon cœur, ma femme Ngọc Diệp, la personne, sans ses encouragements, sa confiance, son soutien constant à travers ces longues années, tout ceci n'aurait pas été possible. Je dédie cette thèse à ma femme, à mon père et aux autres membres de ma famille.

Table des matières

Liste des tableaux

Liste des figures

À ma famille…

Introduction

Le Web sémantique sera-t-il la génération suivante du World Wide Web d'aujourd'hui ?

La capacité de prendre en compte la « sémantique », promise par le futur « Web sémantique », constitue la différence principale entre le Web de demain et celui que l'on connaît de nos jours. Dans le Web sémantique, les informations contenues dans des ressources telles que des pages web, des bases des données, des services… seront disponibles et compréhensibles non seulement pour les hommes mais aussi pour les machines, les programmes ou les agents informatiques. Aujourd'hui, on peut faire beaucoup de choses grâce à l'Internet et au Web. On peut trouver des images satellite de la ville de New York ou savoir la superficie de la Lune ou bien acheter un billet d'avion aller simple Nice-Hanoi, tout cela via le Web. Pourtant on rencontre toujours certaines difficultés quand on a besoin d'un service particulier et un peu plus compliqué. Imaginons que quelqu'un veuille savoir quel est le plus grand cinéma à New York, voir quelques photos satellite de ce cinéma pour avoir une idée de sa situation relative, obtenir le programme des films de ce cinéma dans deux semaines quand cette personne sera à New York pour participer à une conférence scientifique, et supposons que cette personne veuille aussi réserver deux places pour le film de la semaine dans les horaires qui s'adaptent le mieux à son agenda. À l'heure actuelle, pour réaliser le service décrit ci-dessus, on peut se débrouiller d'une manière ou d'une autre. Avec le Web sémantique, demain, on pourrait effectuer des tâches aussi complexes ou voire plus complexes que l'exemple ci-dessus à l'aide d'un programme informatique d'une façon automatique.

Pour que cette belle peinture devienne réalité, le Web sémantique est conçu pour que le contenu des ressources sur le Web puisse être rendu sémantiquement « compréhensible » et accessible par des logiciels. Les ressources disponibles sur l'Internet telles que des documents, des images, des services ou même les ressources physiques qui ne se trouvent pas sur l'Internet mais leurs références y sont disponibles telles que des livres physiques, des personnes… ont une sémantique associée. Grâce à cette sémantique, l'organisation, la sauvegarde, la recherche d'informations pourraient être réalisées, traitées d'une manière automatique par des logiciels.

La sémantique du contenu des ressources dans le Web sémantique doit donc être rendue explicite et disponible pour les machines dans une représentation formelle et standardisée. La standardisation peut aider différents programmes à inter-opérer ou échanger des données. La manière de représenter la sémantique dans le Web sémantique est d'utiliser une *ontologie*. Les objets ainsi que les relations existantes entre eux dans l'univers du discours doivent être conceptualisés si l'on souhaite les représenter pour un but quelconque. Une conceptualisation est une vue abstraite, simplifiée du monde que l'on veut représenter. Une ontologie est alors une spécification explicite d'une conceptualisation [Gruber, 1993]. Une telle spécification comprend des définitions des termes dénotant les entités manipulées dans l'univers du discours ainsi que des textes lisibles pour l'homme décrivant les significations de ces termes, et aussi les axiomes formels qui contraignent l'interprétation et l'utilisation bien formée de ces concepts. L'ontologie est donc employée pour décrire des engagements ontologiques. Si différents agents s'engagent à une ontologie, leurs interactions et leurs communications seront assurées d'une façon logique et cohérente. En d'autres termes, une ontologie correspond à un vocabulaire commun contrôlé, organisé, et partagé, et à la formalisation explicite des relations créées entre les différents termes du vocabulaire.

Problématique

Le web sémantique est une extension du Web actuel, il hérite donc aussi des caractéristiques du Web courant. Une des caractéristiques du Web actuel, qui représente à la fois son point fort et son point faible, est son *hétérogénéité*. Cette caractéristique reflète la vie dans le monde réel où les informations, les connaissances peuvent provenir de plusieurs sources différentes, et être représentées dans des formats divers. De même, dans le Web sémantique, l'hétérogénéité ne sera pas une caractéristique exceptionnelle. Dans ce futur web, les connaissances, les informations peuvent être extraites différemment à partir d'une même ressource par des organisations ou des personnes différentes. Quelqu'un peut dire que la couleur de la voiture du chef d'équipe ACACIA est bleue mais un autre, qui pourrait dire qu'elle est bleue foncé voire qu'elle est verte !

De même qu'au niveau des assertions, ce même désaccord peut se produire au niveau de la conceptualisation. La terminologie choisie dépend de la subjectivité de la personne qui modélise l'ontologie. Il est possible que dans un même domaine, les conceptualisations, donc les ontologies, sont construites différemment. Les termes utilisés dans les vocabulaires peuvent être différents comme ils peuvent être des synonymes. L'un peut employer le terme « voiture » dans une ontologie pour dénoter des objets qui sont des véhicules à quatre roues dans le monde réel, quant à l'autre, qui préfère d'utiliser le terme « bagnole » pour signifier le même objet en question. La différence peut aussi provenir de la granularité dans la spécialisation des concepts. L'un peut considérer que le concept « personne » se divise en deux sous-concepts « homme » et « femme » tandis que l'autre préfère le diviser en trois sous-concepts différents à savoir : « enfant », « adulte » et « personne âgée ».

Tout cela est dû aux préférences des gens, aux différentes catégories d'utilisateurs ou aux différentes sources de connaissances et cela dépend du métier, de la fonction, de la compétence, de l'expérience… de chaque personne. Même dans une organisation, qu'il s'agisse d'une entreprise unique ou d'une entreprise virtuelle constituée par plusieurs organisations en collaboration, on pourra avoir une hétérogénéité au niveau de la terminologie (ou de l'ontologie) ou bien au niveau des assertions.

Cette hétérogénéité posera des problèmes pour l'échange, le traitement, l'intégration, et la recherche d'information.

Objectifs et approche

L'objectif de cette thèse est de permettre la construction et l'exploitation d'un Web sémantique dans une organisation hétérogène, comportant différentes sources de connaissances et différentes catégories d'utilisateurs. Nous étudions des méthodes et des techniques permettant de ne pas éliminer cette hétérogénéité mais de cohabiter avec et d'exploiter cette hétérogénéité dans le Web sémantique.

Dans cette tâche complexe et très difficile, nous nous focalisons sur la partie concernant l'hétérogénéité des ontologies. Cette partie se divise en deux sous-problèmes : (1) Faire la correspondance entre deux ontologies différentes, tâche qui va permettre l'interaction et la communication entre les agents ainsi que l'échange, l'intégration et la recherche d'information ; (2) aider à construire une ontologie multi-points de vue pour une organisation, ce qui permettra d'héberger des terminologies et des engagements ontologiques en respectant toujours l'hétérogénéité, des points de vue de différents ontologistes ainsi de différentes sources de connaissance (e.g. différents experts, différents annotateurs).

Plan du Mémoire

Comme de coutume, ce mémoire commence par un premier chapitre dédié à l'état de l'art. Dans un premier temps, nous présentons (i) des notions de base essentielles dans notre contexte de recherche telles que la mémoire d'entreprise, le Web sémantique, le Web sémantique d'entreprise, l'ontologie, le point de vue... (ii) l'analyse des différences entre le Web sémantique et le web courant, (iii) l'architecture du Web sémantique et les briques de base du Web sémantique.

Toujours dans le chapitre 1, nous détaillons ensuite la problématique de l'hétérogénéité dans le Web sémantique et nous présentons notre solution. Nous présenterons des travaux existants reliés à cette problématique : les travaux sur l'alignement des ontologies. Nous présenterons aussi les travaux qui intègrent la notion de point de vue dans la représentation des connaissances.

Dans le chapitre 2, nous présentons un algorithme permettant de faire la correspondance entre des ontologies représentées en `RDF(S)`. Les caractéristiques de ces types d'ontologie sont rendues explicites pour que l'on puisse les utiliser dans la tâche d'alignement des ontologies en `RDF(S)`. L'algorithme proposé, nommé ASCO1, exploite donc les informations disponibles dans les ontologies en `RDF(S)` telles que les étiquettes, les commentaires, la structure des ontologies…pour calculer les valeurs de similarité entre des éléments dans les ontologies. Ces valeurs sont ensuite utilisées pour évaluer les correspondances entre deux ontologies. Parmi les techniques utilisées, la technique appliquant la fréquence des termes TF/IDF montre un des points forts de l'algorithme.

Le langage recommandé par le W3C pour des ontologies dans le Web sémantique `OWL` est considéré comme une extension du langage `RDF(S)`. Il fournit en plus de nouveaux constructeurs avec une sémantique bien définie. Le chapitre 3 est dédié à

l'étude des méthodes exploitant ces constructeurs et leur sémantique pour aligner des ontologies représentées en OWL. Cette fois, la structure de l'ontologie ainsi que les primitives de OWL sont utilisées dans l'algorithme ASCO2 pour calculer la similarité entre deux éléments dans deux ontologies représentées en OWL. L'algorithme ASCO3 va plus loin dans la comparaison de la similarité de deux graphes représentant les structures de l'ontologie. Il applique l'algorithme de recherche du sous-graphe commun maximal de deux graphes, qui se base sur la recherche de la clique maximale du graphe associé (une clique est un sous-graphe complet dont les sommets sont tous connectés deux à deux).

Une évaluation des algorithmes d'alignement d'ontologies proposés (ASCO1, ASCO2 et ASCO3) est présentée dans le chapitre 4. L'implémentation des algorithmes a été effectuée en Java et les évaluations ont été faites (1) sur notre base de tests, qui se compose de deux ontologies en RDF(S) nommées O'CoMMA et O'Aprobatiom; et (2) sur des bases de tests standard publiées à http://www.atl.external.lmco.com/projects/ontology/ et http://oaei.ontologymatching.org/.

Le cinquième chapitre précise comment nous avons construit un modèle pour représenter des ontologies avec plusieurs points de vue. Ce modèle est ensuite représenté dans le langage MVP-OWL que nous avons proposé. Ce langage est une extension du langage OWL, recommandé par le W3C pour représenter des ontologies dans le Web sémantique. Il nous permet d'écrire des ontologies multi-points de vue selon le modèle MVP que nous avons proposé. Ensuite, la gestion de multiples points de vue, l'exploitation des points de vue pour la recherche d'information dans la base d'annotations sont étudiées. Enfin, une méthode est proposée pour guider les cogniticiens à construire une nouvelle ontologie multi-points de vue à partir d'ontologies déjà existantes avec l'aide des alignements trouvés par les algorithmes ASCO1, ASCO2 ou ASCO3 ou par d'autres algorithmes d'alignement d'ontologies.

Avant la conclusion, dans le chapitre 6, nous présentons une illustration du modèle de représentation des connaissances multi-points de vue MVP et du langage d'ontologie multi-points de vue MVP-OWL proposés dans le chapitre 5.

En conclusion, nous présentons un bilan de notre travail et nous analysons nos principales contributions. Nous discutons également des perspectives de ce travail.

Chapitre 1

État de l'art

Dans ce chapitre, nous présentons des connaissances de base dans notre domaine de recherche. Il s'agit des principes et des technologies du Web sémantique. Ensuite, nous montrons des techniques de base pour calculer la similarité entre deux entités dans le monde du Web sémantique, la base pour résoudre le problème de l'hétérogénéité. Nous présentons aussi des travaux dans la littérature qui attaquent ce difficile problème. Enfin, nous présentons la notion de point de vue dans le Web sémantique et des approches employant cette notion dans la gestion des connaissances ou pour fournir des services évolués dans le Web sémantique.

1.1 Le Web sémantique

Le Web que nous connaissons et utilisons aujourd'hui est dédié à l'utilisation pour des êtres humains. Dans ce Web, les gens peuvent créer des pages Web, écrire ce qu'ils veulent dire, dessiner et décorer comme ils imaginent, mettre en page un document comme ils le souhaitent. Le but est de fournir des informations, des contenus aux utilisateurs humains et de laisser ceux-ci interpréter ces contenus. Quant aux logiciels, ils ne comprennent rien, leur mission est de représenter les informations, les contenus exactement comme leurs auteurs souhaitent. Ils ne sont pas capables de comprendre qu'une page web donnée concerne une personne, qu'une autre page décrit les caractéristiques d'une nouvelle voiture de Renault. Il est impossible de demander à la machine de trouver de manière automatique le nom de la personne qui a conçu ce modèle de voiture (bien que quelqu'un puisse toujours se débrouiller pour le faire de façon ou d'autre, de manière manuelle mais le résultat dépend du niveau de compétence de la personne et le temps et l'énergie qu'il doit dépenser ne sont pas toujours raisonnables).

Le but du Web sémantique est de rendre explicite le contenu sémantique des ressources dans le Web (documents, pages web, services…). Les ordinateurs et les agents logiciels pourraient donc « comprendre » les informations contenues dans ces ressources et aider les utilisateurs à exécuter et compléter leurs tâches, leurs requêtes de façon plus automatique et plus efficace. Les machines maintenant pourraient « savoir » qu'une voiture est une sorte de véhicule, que c'est la même chose qu'une bagnole, que l'âge d'une personne est un nombre entier positif et ne peut pas être plus grand que 150 ans… Elles pourraient donc raisonner sur des connaissances déjà existantes dans le Web pour fournir des informations plus précises, des services plus évolués qui s'adaptent mieux aux besoins des utilisateurs.

Cependant, le Web sémantique n'est pas construit à partir de zéro. Le Web actuel est déjà un trésor immense des connaissances et des informations dans l'histoire humaine, où les informations sont sauvegardées, organisées et représentées de façon passive, non structurée, arbitraire et ad hoc. Le Web sémantique hérite de ce trésor des connaissances mais il permet aussi à des machines d'accéder à ce trésor et de l'exploiter. Le Web sémantique est donc une extension du Web actuel.

La Figure 1 montre en image les extensions du Web sémantique par rapport au Web actuel. Nous pouvons trouver, en plus des ressources dans le Web actuel, leur sémantique rendue explicite dans le Web sémantique, souvent sous forme d'annotations sémantiques de ces ressources. Les pages web codées en HTML (HyperText Markup Language, langage pour mettre en page des documents, des textes, des images, des vidéos… pour les pages web), transférées sur le Web par le protocole de communication HTTP (HyperText Transfer Protocol), référencées (la localisation) par des URL (Uniform Resource Locator) dans le Web actuel seront identifiées par des URI (Uniform Resource Identifier) dans le Web sémantique. En plus, la sémantique des ressources sera encodée selon le modèle de méta-données standardisé `RDF` (Resource Description Framework), sérialisées en format XML (Extensible Markup Language) et basées sur des ontologies en format `OWL` (Web Ontology Language). Dans le Web sémantique, les ressources seront ainsi accessibles et compréhensibles par des logiciels, qui pourront effectuer des

raisonnements, des recherches plus « intelligentes » que les recherches par des mots clés comme aujourd'hui.

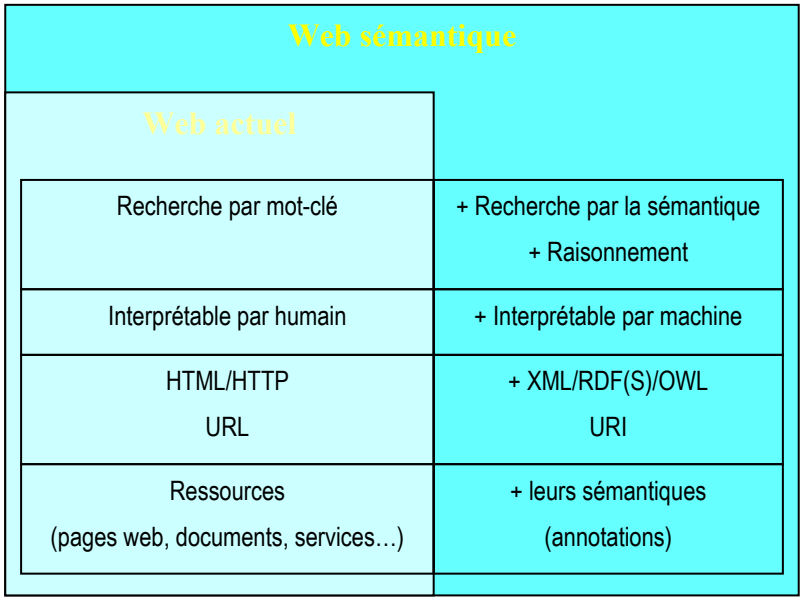


Figure 1 Le Web actuel et son extension, le Web sémantique

Comme nous l'avons indiqué ci-dessus, la sémantique des ressources dans le Web sémantique est rendue explicite et représentée sous forme d'annotations sémantiques. L'interprétation des annotations, donc la sémantique, est précisée entre des agents logiciels, des machines ou voire des gens grâce à un des éléments les plus importants du Web sémantique : *l'ontologie*. Avant de présenter l'architecture du Web sémantique (*1.1.2*), des langages pour le Web sémantique (*1.1.3*) et le Web sémantique d'entreprise (*1.1.4*), nous étudions la notion d'ontologie dans la section suivante pour voir son rôle important dans le monde du Web sémantique.

1.1.1 Ontologies

Les ontologies sont étudiées par les chercheurs travaillant en intelligence artificielle, sur la représentation de la connaissance, et maintenant, sur le Web sémantique. Une définition des ontologies consensuelle, précise et complète dans le contexte du Web sémantique n'existe pas encore. Cependant, la définition la plus citée dans notre domaine de recherche est celle de [Gruber, 1993]: « *Une ontologie est une spécification explicite et formelle d'une conceptualisation d'un domaine de connaissance* ». La *conceptualisation* est le résultat d'une analyse ontologique du domaine étudié et donc est l'abstraction du monde de ce domaine. Cette conceptualisation est représentée dans une forme concrète, donc *spécification*, où les concepts, les relations entre eux et les contraintes pour les employer sont *explicitement* définis dans un format et langage *formel*.

[Neches *et al.*, 1991] présente la première définition de l'ontologie dans le domaine de l'informatique : « Une ontologie définit les termes et les relations de base comportant le vocabulaire d'un domaine aussi bien que les règles pour combiner des termes et les relations afin de définir des extensions du vocabulaire ».

Dans [Guarino et Giaretta, 1995], les auteurs essaient de donner une clarification terminologique du terme « ontologie » utilisé dans le domaine de l'intelligence artificielle. Les sept interprétations possibles du terme « ontologie » sont analysées en se basant sur la notion de la « *conceptualisation* » qui est définie dans une manière sémantique rigoureuse. Enfin, une ontologie est considérée comme « *une théorie logique qui donne une explication explicite et partielle d'une conceptualisation* » (sens 1) ou « *synonyme de la conceptualisation* » (sens 2).

Les auteurs de [Paolo *et al.*, 2003] définissent les ontologies comme « des modèles partagés d'un domaine encodant une vue qui est commune à un ensemble de différentes parties »

Néanmoins, la plupart des chercheurs sont d'accord pour considérer que la notion d'ontologie est dérivée de celle de la philosophie où on étudie l'être ou l'existence aussi bien que ses catégories de base en essayant de découvrir quelles entités, quels types d'entités et quelles relations entre des entités existent.

Dans notre contexte, nous considérons une ontologie d'un domaine de connaissance informellement comme un vocabulaire commun des termes, qui est bien contrôlé, organisé, cohérent, et partagé. Ces termes correspondent aux concepts et relations entre eux dans le domaine. Les termes sont organisés dans une structure hiérarchique formée par des liens de « *spécialisation* » (subsomption ou is_a) entre des termes de concepts ou entre des termes de relations. Ce vocabulaire commun est partagé entre des agents qui collaborent dans le domaine en question. Cela assure une interprétation cohérente, précise entre les agents.

Le problème de l'hétérogénéité dans le Web se produit aussi dans le Web sémantique, donc pour les ontologies. [Visser *et al.* 1997] présente une analyse et une classification des disparités (mismatches) entre des ontologies : disparité au niveau de la conceptualisation et au niveau de l'explication. Pour chaque niveau, il détaille de différents types de disparité tels que la disparité des classes, des relations, des structures… et étudie le traitement de chaque type de disparité. [Hameed *et al*. 2003] approfondit des problèmes et techniques pour la réconciliation des disparités entre des ontologies. Il propose un processus de réconciliation des ontologies. Enfin, il présente une revue des outils qui supportent la réconciliation tels que Chimæra [McGuinness *et al.*, 2000], ONION [Mitra *et al.*, 2000], OntoView [Klein *el al.*, 2002] et Prompt [Noy et Musen, 2001].

1.1.2 L'architecture du Web sémantique

Dans cette section, nous présentons brièvement les composants de base sur lesquels on construit le Web de demain, le Web sémantique (*Figure 2*).

Le premier facilitateur pour les technologies du Web sémantique est l'*URI* (Uniform Resource Identifier - identifiant uniforme de ressource). L'URI est employé pour désigner le nom ou l'adresse (ou les deux) d'une ressource dans le Web sémantique sans distinction d'une ressource physique (donc sa représentation est récupérable via l'Internet telle qu'une page web, un service localisé sur un serveur…) ou d'une ressource abstraite (un livre particulier, une idée…). Rappelons que l'URL (Uniform Resource

Locator), comme l'URI, est une chaîne courte des caractères mais il n'est employé que pour référer des ressources (physiques) par leur localisation. Notons aussi que quelque chose qui peut être identifié avec un URI peut être décrit, ainsi le Web sémantique peut raisonner au sujet des personnes, des endroits, des idées… Les URIs sont utilisés pour identifier (nommer) des ressources, mais sans procédé pour les récupérer.

Quelques exemples d'URI :

- http://www.inria.fr/acacia/index.htm (pour une page web)
- http://www.inria.fr/acacia/OntologyMatching.pdf (pour un article)
- rstp://www.video.com/france.rm (pour une vidéo)
- urn:issn:2242-4157 (pour un livre)
- tel:+33-497-15-53-17 (pour un numéro de téléphone)

XML (Extensible Markup Language) fournit une syntaxe pour les documents structurés (dans un structure hiérarchique, un arbre), mais n'impose aucune contrainte sémantique à la signification de ces documents. Un langage à balises combine le texte et les informations supplémentaires (méta-données) sur le texte. Les informations supplémentaires, telles que la structure, la police, la couleur… des textes, sont exprimées en utilisant des balises, qui sont mélangées avec le texte à présenter. En plus, le langage XML permet aux utilisateurs de définir eux-mêmes leurs balises.

Un *schéma XML* est une description du type d'un document XML qui contient un ensemble de règles (contraintes sur la structure et le contenu du document) auxquelles un document XML doit se conformer afin d'être considéré valide selon ce schéma. DTD (Document Type Definition), RELAX NG (REgular LAnguage for XML Next Generation), XML Schema sont spécifiquement développés pour exprimer des schémas de XML.

L'*espace de noms* (namespace) est un contexte ou un conteneur abstrait contenant des noms, des termes, des mots qui représentent des objets, des concepts dans le monde réel. Un nom défini dans un espace de noms correspond à un et seulement un objet, deux objets ou concepts différents sont référencés par deux noms différents dans un même espace de noms.

Voici un exemple du schéma de XML pour décrire un article :

```
<xs:schema
 xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="article">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="title" type="xs:string"/>
    <xs:element name="numOfPages" type="xs:integer"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

L'exemple de document XML ci-dessous se conforme à ce schéma :

```
<article
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="article.xsd">
  <title>Ontology matching</title>
  <numOfPages>15</numOfPages>
</article>
```

Dans les exemples ci-dessus, `xs`, `xmlns`, `xsi` sont des espaces des noms.

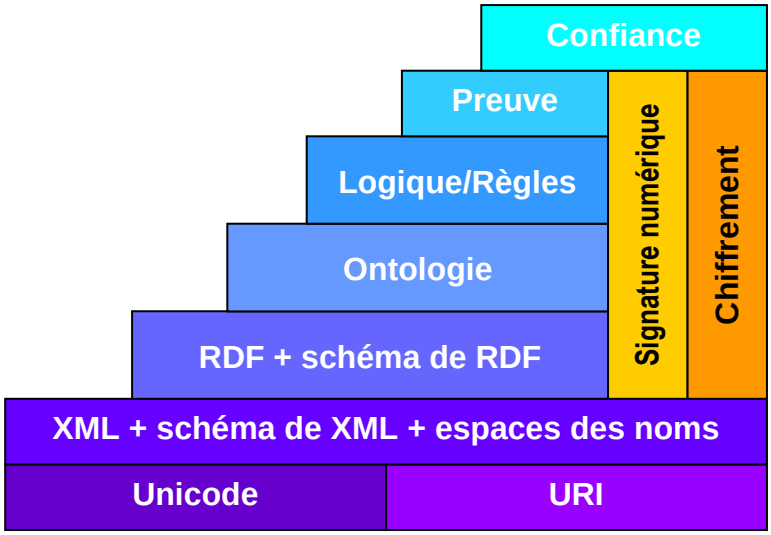


Figure 2 l'architecture du Web sémantique

RDF (Resource Description Framework) est un modèle de méta-données pour référencer des objets (ressources) et comment ils sont reliés l'un à l'autre. Dans ce modèle, les ressources sont identifiées (référencées) par les URIs et nous pouvons faire des déclarations à propos de ces ressources en employant des expressions sous forme « sujet – prédicat – objet », appelées des triplets. Le sujet est la ressource, la « chose » à décrire. Le prédicat est un attribut ou un aspect de cette ressource, et exprime un lien ou une relation entre le sujet et l'objet. L'objet est l'objet de la relation ou de la valeur de ce prédicat. Les triplets `RDF` peuvent se décrire en XML.

La *Figure 3* montre un exemple d'un modèle RDF :

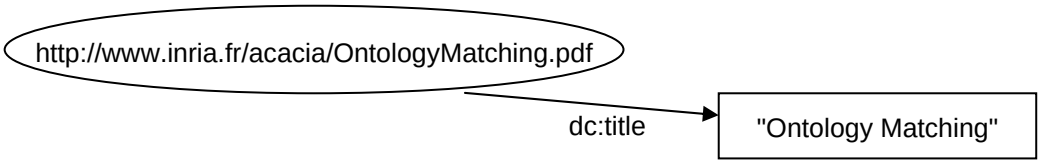


Figure 3 Un exemple d'un modèle RDF

Sa représentation sous forme de triplet est :

```
<http://www.inria.fr/acacia/OntologyMatching.pdf> <dc:title> "Ontology Matching" .
```

Et sa représentation en syntaxe XML :

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/dc/elements/1.1/">
  <rdf:Description rdf:about="http://www.inria.fr/acacia/OntologyMatching.pdf">
      <dc:title>Ontology Matching</dc:title>
  </rdf:Description>
</rdf:RDF>
```

RDF Schema est un langage pour décrire des vocabulaires, des propriétés et des classes de ressources dans le modèle RDF. RDF Schema est une extension sémantique de RDF. Il fournit des mécanismes pour décrire des groupes de ressources similaires (classes) et des relations entre ces ressources (propriétés). Les descriptions de vocabulaire de RDF Schema sont écrites en RDF en utilisant les termes (primitives) décrits dans la spécification du schéma RDF[1]. La combinaison de RDF Schema et RDF est souvent référencée par RDF(S). Autrement dit, RDF(S) fournit un moyen pour décrire les types des ressources et leurs caractéristiques.

Dans l'exemple suivant, nous employons des primitives du schéma RDF (tels que rdfs:Class, rdfs:label…) pour définir la classe « Article » (un groupe d'articles scientifiques) et décrire des caractéristiques de cette classe, donc des caractéristiques des instances (ressources) appartenant à cette classe, telles que leur titre, leur nombre de pages. Toutes les descriptions sont écrites en RDF.

```
<rdf:RDF
  xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:xsd ="http://www.w3.org/2001/XMLSchema#"
  xml:base  ="http://www.inria.fr/acacia/exemple#">

  <rdfs:Class rdf:about="Article">
      <rdfs:label xml:lang="en-US">l'article scientifique</rdfs:label>
  </rdf:Class>
  <rdfs:Property rdf:about="title">
      <rdfs:label xml:lang="en-US">titre de l'article scientifique</rdfs:label>
      <rdfs:domain rdf:resource="Article"/>
      <rdfs:range rdf:resource="&xsd;string"/>
  </rdf:Class>
  <rdfs:Property rdf:about="pageNum">
      <rdfs:label xml:lang="en-US">nombre de pages</rdfs:label>
      <rdfs:domain rdf:resource="Article"/>
      <rdfs:range rdf:resource="&xsd;integer"/>
  </rdf:Class>
</rdf:RDF>
```

Une des caractéristiques les plus importantes RDF Schema est que nous pourrons définir les liens de « subsomption » entre des classes et des relations en employant les primitives rdfs:subClassOf et rdfs:subPropertyOf. Ce sont les liens de « spécialisation » ou « is_a » qui permettent aux classes et aux relations d'hériter des caractéristiques définies dans des classes (ou des relations) parentes. Cela permet des raisonnements dans RDF(S).

OWL (Web Ontology Language) est un langage pour représenter des ontologies dans le Web sémantique. C'est une extension du vocabulaire de RDF(S). OWL est dérivé du langage d'ontologie DAML+OIL[2]. En comparaison avec RDF(S) pour décrire des

[1] http://www.w3.org/TR/rdf-schema/

[2] http://www.daml.org/2001/03/daml+oil-index

propriétés et des classes, OWL permet en plus d'exprimer, entre d'autres, des relations entre des classes (telles que la disjonction), la cardinalité (par exemple "exactement un"), l'égalité, 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. Une ontologie est capable de décrire des rapports entre des types de choses mais ne contient pas n'importe quelles informations indiquant comment employer ces rapports dans les calculs.

Voici un exemple d'une ontologie en OWL :

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

  <owl:Class rdf:ID="Animal">
    <rdfs:label>Animal</rdfs:label>
    <rdfs:comment>
      This class of animals is illustrative of a number of ontological idioms.
    </rdfs:comment>
  </owl:Class>
  <owl:Class rdf:ID="Person">
    <rdfs:subClassOf rdf:resource="#Animal"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasParent"/>
        <owl:allValuesFrom rdf:resource="#Person"/>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:ObjectProperty rdf:ID="hasAncestor">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:domain rdf:resource="#Animal"/>
    <rdfs:range rdf:resource="#Animal"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="hasParent">
    <rdfs:subPropertyOf rdf:resource="#hasAncestor"/>
  </owl:ObjectProperty>
</rdf:RDF>
```

Dans l'architecture du Web sémantique (*Figure 2*, inspirée de l'architecture proposée par Berners-Lee dans [Berners-Lee, 2000]), nous voyons les liens entre des éléments du Web sémantique présentés ci-dessus. L'URI, l'Unicode, XML sont considérés comme des briques de base, sur lesquelles reposent des langages tels que RDF(S) permettant de décrire des « choses » (objets, concepts…) et leurs types ; puis des langages tels que OWL pour exprimer des relations entre des types des choses mais sans spécifier comment exploiter ces informations dans les calculs. En combinant avec la logique, les règles, des assertions autour du Web (stockées par des langages des couches inférieures tels que RDF(S), OWL) peuvent être employées pour déduire de nouvelles connaissances. Au-dessus de la couche de logique, c'est la couche de preuve. Ici, nous avons un langage universel pour représenter des preuves, qui permet à différents systèmes de partager leurs connaissances « originales » ou déduites. La provenance (l'origine) 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 employé. Enfin, le but final vers lequel nous nous orientons est la confiance, un Web sémantique et fiable, où nous pouvons effectuer plusieurs tâches complexes en sûreté.

1.1.3 Les langages du Web sémantique

Dans les sections précédentes, nous avons examiné des technologies, des éléments essentiels du Web sémantique tels que l'ontologie, le modèle des données. Pour que ces technologies deviennent utiles et traitables par des machines, elles doivent être exprimées en des notations concrètes en utilisant des langages formels. Dans cette section, nous présentons les langages utilisés dans le Web sémantique.

Comme présenté dans la section *1.1.2*, le langage XML est la base, sur laquelle nous construisons d'autres langages. En fait, XML est vraiment un métalangage pour décrire des langages à balises. Il n'indique ni la sémantique des balises, ni un ensemble prédéfini des balises. En d'autres mots, XML permet de définir des balises et des liens structurels entre elles.

1.1.3.1 Les langages, les formalismes pour représenter des assertions

John F. Sowa a proposé le formalisme des graphes conceptuels (GC) dans [Sowa, 1984] comme un système de logique reposant sur les graphes existentiels (qui permettent de représenter visuellement des expressions logiques) et sur les réseaux sémantiques. Les graphes conceptuels permettent d'exprimer la signification, les faits sous une forme qui est logiquement précise, et à la fois lisible pour les gens, et plus facile à être traitée par la machine. Les GCs ont été mis en application dans plusieurs domaines de recherche tels que la recherche d'information, les systèmes experts, le traitement de langage naturel et le Web sémantique. Avec ce formalisme, nous pouvons représenter des connaissances et effectuer des raisonnements en employant des opérateurs définis dans le formalisme tels que l'opérateur de projection.

La *Figure 4* montre un graphe conceptuel avec quatre concepts: [`Allez`], [`Acaciens`: ∀], [`Lieu`: `Bureau`], et [`Voiture`]. Ce graphe comprend trois relations conceptuelles : (`Agnt`) relie [`Aller`] à tous les [`Acaciens`], (`Dest`) relie [`Aller`] au lieu [`Sophia`], et (`vhcl`) relie [`Aller`] à la [`Voiture`]. La connaissance représentée ici donc est « tous les acaciens vont à Sophia en voiture ».

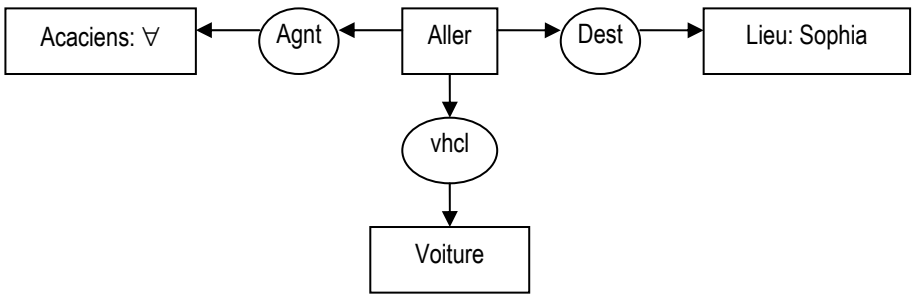


Figure 4 Un exemple d'un graphe conceptuel

Le modèle pour représenter des méta-données `RDF` permet aussi de représenter des connaissances en déclarant des expressions sous forme de « sujet-prédicat-objet » des ressources (objets réels, concepts…). Tandis que les GCs permettent de représenter des relations n-aire, `RDF` ne permet que d'exprimer des relations binaires. Cependant, les relations d'arité plus élevée peuvent toujours être représentées dans `RDF` en employant la

technique suivante : créer une classe qui représente la relation n-aire et n attributs pour la nouvelle classe correspondant à chaque arguments de cette relation. RDF n'a pas de mécanismes pour définir des relations entre des propriétés et entre des ressources, c'est le rôle de RDF Schema. La combinaison de ces deux technologies, RDF(S), permet de représenter des connaissances non seulement au niveau des assertions (des faits) mais aussi au niveau de la conceptualisation (niveau d'ontologie). La sémantique prédéfinie dans RDF(S) permet de raisonner et de déduire de nouvelles connaissances. Les documents en RDF(S) sont souvent sérialisés (écrits) en XML. Cela permet d'exploiter plusieurs outils déjà existants pour XML en traitant les documents en RDF(S). Pour plus de lisibilité, d'autres langages sont aussi proposés pour représenter les modèles RDF(S) dans des formats tels que Notation 3 (N3)[1], NTriples, Turtle…

Alors que RDF(S) est une recommandation de W3C (World Wide Web Consortium), Topic Maps est une norme de l'ISO (International Organization for Standardization) pour la représentation et l'échange de connaissances, avec une emphase sur la capacité de recherche d'information. Comme RDF(S), Topic Map permet de représenter des informations, des connaissances en employant des sujets (topics) (pour représenter des concepts, des personnes, des événements…), des associations (pour représenter des rapports entre des sujets), et des occurrences (pour représenter des rapports entre des sujets et des ressources d'information qui les concernent). Donc, les sujets dans Topic Map correspondent aux classes de RDF(S), les associations aux relations de RDF(S) et les occurrences aux spécialisations (instancialisation) dans RDF(S). Plusieurs travaux visent à offrir l'interopérabilité entre ces technologies [rdftm-survey, 2005] [Garshol, 2003].

1.1.3.2 Les langages pour représenter des ontologies

Un langage d'ontologie est un langage formel permettant de représenter une ontologie. Comme nous l'avons indiqué dans la section précédente, nous pouvons utiliser RDF(S) pour représenter des ontologies simples. Les ontologies en RDF(S) peuvent être sérialisées en des langages tels que XML ou N3. Cependant, une ontologie est employée pour représenter des connaissances dans un domaine, donc il est nécessaire de disposer d'un langage aussi expressif pour les représenter, et RDF(S) ne répond pas à ces besoins. Par exemple, en utilisant RDF(S), on ne peut pas représenter la cardinalité d'une relation ou exprimer des caractéristiques des relations telles que la transitivité, la symétrie ou la fonctionnalité, ou de faire des restrictions pour certaines classes… Ainsi, le W3C a recommandé un langage standardisé plus puissant au niveau expressivité, qui est spécialement conçu pour représenter des ontologies dans le Web sémantique. Cela permet avec facilité de créer, partager et échanger des connaissances dans le Web sémantique. Le langage d'ontologie recommandé est le langage OWL. Il est dérivé du langage DAML+OIL[2]. OWL couvre la plupart des caractéristiques du langage DAML+OIL et renomme la plupart de ses primitives.

Le langage d'ontologie OWL est divisé en trois sous-langages avec une puissance d'expressivité dégressive : OWL Full, OWL DL et OWL Lite. La raison de cette division

[1] http://www.w3.org/DesignIssues/Notation3.html

[2] http://www.daml.org/2001/03/daml+oil-index

concerne la complexité, la calculabilité et l'implémentation du langage. Tandis que le sous-langage OWL Lite a la complexité formelle la plus basse, l'expressivité minimale dans la famille, il est suffisant pour représenter des thésaurus et d'autres taxonomies ou des hiérarchies de classification avec des contraintes simples. Les outils à implémenter pour supporter OWL Lite et les migrations pour les thésaurus, les taxonomies déjà existants sont aussi moins coûteux. Quand à OWL DL, il a une expressivité maximum tout en maintenant la complétude computationnelle (toutes les conclusions sont garanties pour être calculées) et la décidabilité (tous les calculs finiront dans le temps fini). OWL DL correspond à la variante de la logique de description $\mathcal{SHOIN}(\mathcal{D})$ [Horrocks *et al*., 2003]. Le sous-langage OWL DL est donc approprié pour représenter des ontologies ayant besoin de la puissance d'expressivité tout en gardant la calculabilité. Enfin, OWL Full est conçu pour les développeurs, les implémenteurs et les utilisateurs qui ont besoin de l'expressivité maximale, de la liberté syntaxique de RDF mais sans garantie de calculabilité. Actuellement, il n'existe pas encore des outils ou des logiciels de raisonnement qui sont capables de supporter des raisonnements complets pour toutes les caractéristiques de OWL Full.

En ce qui concerne de la compatibilité de ces sous-langages, le sous-langage OWL Full peut être considéré comme une extension de RDF, tandis que OWL Lite et OWL DL peuvent être considérés comme des extensions d'une vue restreinte de RDF. OWL Full est une extension de OWL DL, et ce dernier est une extension de OWL Lite. Une ontologie légale en OWL Lite est aussi légale en OWL DL et OWL Full. Tous les documents en OWL (Full, DL, Lite) sont des documents valides en RDF, et un document RDF est un document OWL Full, mais seulement quelques documents en RDF sont des documents légaux en OWL Lite ou OWL DL.

1.1.4 Web Sémantique d'Entreprise

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. Pour faciliter et améliorer l'accès, le partage, la réutilisation de cette mémoire d'entreprise, voire permettre la création ou la déduction des nouvelles connaissances à partir de celle-ci, il est nécessaire de disposer des moyens et des outils permettant de matérialiser la mémoire et d'indexer les contenus dans cette mémoire d'entreprise [Dieng *et al.*, 2004].

En faissant l'analogie entre les ressources d'une mémoire d'entreprise et celles du Web, l'on peut employer les technologies du Web pour la gestion, la représentation et l'exploitation des connaissances dans une organisation : l'implémentation et l'utilisation de l'Intranet au sein de l'organisation. Avec l'évolution du Web vers le Web sémantique, les nouvelles technologies pour ce dernier peuvent aussi être appliquées pour l'organisation : les mémoires d'entreprise sont construites sous forme du *Web sémantique d'entreprise* [Dieng *et al.*, 2004]. Les technologies du Web sémantique présentées dans les sections précédentes sont employées pour réaliser le Web sémantique d'entreprise : les annotateurs annotent sémantiquement des ressources d'entreprise : description du contenu des documents, compétences des personnes, caractéristiques des services,… en employant le vocabulaire commun et partagé au sein de l'entreprise (l'ontologie de cette entreprise). Les utilisateurs dans l'entreprise, grâce à cette ontologie commune, pourront

alors interpréter des annotations créées par des annotateurs grâce aux significations des termes, des concepts bien prédéfinies dans l'ontologie. Cela permet aux anciens utilisateurs de partager leurs connaissances, leurs expériences, leurs compétences et aux nouveaux utilisateurs d'apprendre, de consulter et de s'adapter mieux à l'entreprise. Cela facilite les utilisateurs dans leurs tâches individuelles et collectives.

1.2 Alignement d'Ontologies

Comme nous l'avons indiqué, le langage `OWL` recommandé par le W3C pour représenter les ontologies dans le Web sémantique est construit à partir du langage `RDF(S)` qui est aussi recommandé par W3C, en ajoutant des nouvelles primitives dont la sémantique est précisée. Le langage `OWL` peut donc être considéré comme une sorte d'extension du langage `RDF(S)`. Mais `OWL Lite` et `OWL DL` ne sont pas une *vraie* extension de `RDF(S)` : toutes les définitions en `RDF(S)` ne sont pas toujours des définitions valides en `OWL Lite/DL` [Grosof *et al*., 2003]. Les algorithmes de recherche des correspondances ou d'alignement des ontologies représentées en `RDF(S)` pourraient aussi être appliqués (avec peu de modifications) pour les ontologies représentées en `OWL`.

Les ontologies sont très souvent représentées par des hiérarchies taxonomiques des classes et des relations, bien qu'elles n'aient pas besoin d'être limitées à ces formes. Les liens entre des classes ou entre des relations dans ces taxonomies sont les liens de subsomption. Une ontologie simple, qui n'offre pas de relations entre classes (autre que ce lien de subsomption) et donc est sans hiérarchie des relations, peut être considérée comme un schéma. Du fait de cette observation, il est possible d'adapter des algorithmes d'alignement des schémas de base de données, qui ont déjà été bien étudiés dans le domaine des bases de données, dans le contexte de l'intégration de données et de la traduction de données, à l'alignement des ontologies dans le contexte de la représentation des connaissances.

Dans cette partie, nous allons examiner les techniques et les méthodes utilisées dans la littérature qui attaquent le problème de recherche de la similarité, de la dissimilarité ou de la correspondance entre deux entités en général, qu'elles apparaissent dans des schémas, ou dans des ontologies représentées soit en `RDF(S)`, soit en `OWL`. Ensuite, des approches qui emploient ces techniques seront présentées. Enfin, un tableau récapitulatif résumera une vue globale des approches et leurs relations avec des techniques présentées.

Remarque : La structure de cette partie et quelques méthodes présentées ici sont inspirées et résumées de revue [Euzenat *et al*., 2004], qui est une revue des approches d'alignement des ontologies, réalisée par plusieurs équipes et laboratoires européens dans le réseau d'excellence « Knowledge Web »[1], auquel nous avons participé.

[1] http://knowledgeweb.semanticweb.org/

1.2.1 Les méthodes de base pour mesurer la similarité

Avant de présenter ces méthodes, nous donnons la définition de la similarité.

1.2.1.1 La similarité

La notion de similarité dans notre contexte n'est pas celle que l'on peut trouver en psychologie ou en mathématiques. En psychologie sociale, la similarité se rapporte à comment les attitudes, les valeurs, les intérêts et la personnalité correspondent entre les personnes.

En mathématiques, plusieurs relations d'équivalence (qui sont des relations binaires réflexives, symétriques et transitives) sont appelées la similarité. Ces relations existent par exemple :

(i) En géométrie : Deux objets géométriques sont similaires si l'un est isométrique avec le résultat d'un agrandissement ou rétrécissement uniforme de l'autre. L'un peut être obtenu à partir de l'autre uniformément élargi, rétréci, avec une rotation éventuelle (tous les deux ont la même forme), ou en plus, en appliquant un effet du miroir (l'un a la même forme que l'image de miroir de l'autre). Par exemple, tous les cercles sont similaires entre eux, tous les carrés sont similaires entre eux, et toutes les paraboles sont similaires entre elles. D'autre part, ni les ellipses, ni les hyperboles ne sont similaires entre elles. Deux triangles sont semblables si et seulement si ils ont les mêmes 3 angles.

(ii) En algèbre linéaire, deux matrices A et B de taille $n \times n$ sont dites similaires s'il existe une matrice inversible P de même taille $n \times n$ satisfaisant $P^{-1}AP = B$.

(iii) En topologie, la similarité est une fonction telle que sa valeur est plus grande quand deux points sont plus proches (contrairement à la distance, qui est une mesure de dissimilarité : plus les points sont proches, plus la distance est petite).

Dans notre contexte, la notion de similarité est plutôt celle de la similarité sémantique, qui est également appelée la *proximité sémantique*. Elle est déterminée grâce à l'association à des documents, des termes ou des entités, d'une métrique basée sur la similitude de leurs significations ou de leurs contenus sémantiques. *La similarité est la quantité qui reflète la force du rapport entre deux objets ou deux caractéristiques*. Concrètement, ceci peut être réalisé par exemple en définissant une similarité topologique, ou en employant des ontologies pour définir une distance entre les termes (une métrique naïve pour des termes représentés comme noeuds dans une hiérarchie ontologique pourrait être la distance minimale des arcs différents entre les deux noeuds), ou en employant des moyens statistiques pour faire corréler des mots et des contextes textuels.

Dans la plupart des approches dans notre contexte, la notion de similarité sémantique est vue comme celle de la similarité topologique en mathématiques, où on l'associe à une fonction, appelée *fonction de la similarité*. La définition de cette fonction de la similarité peut changer selon les approches, selon les propriétés souhaitées. La valeur de cette fonction est souvent comprise entre 0 et 1, ce qui permet des possibilités d'interprétation probabiliste de la similarité. Des propriétés ou des caractéristiques

communes possibles de la fonction sont des caractéristiques positives, auto similaires ou maximales, symétriques ou réflexives. On peut aussi trouver d'autres caractéristiques telles que la finitude ou la transitivité.

> ***Définition 1*** *(Similarité). La similarité S : O × O → R est une fonction d'une paire d'entités à un nombre réel exprimant la similarité entre ces deux entités telle que:*
>
> - $\forall a, b \in O, S(a, b) \geq 0$ *(positivité)*
> - $\forall a, b, c \in O, S(a, a) \geq S(b, c)$ *et* $S(a, a) = S(a, b) \Leftrightarrow a = b$ *(auto-similarité ou maximalité)*
> - $\forall a, b \in O, S(a, b) = S(b, a)$ *(symétrie)*
> - $\forall a, b, c \in O, S(a, b) = S(b, c) \Rightarrow S(a, b) = S(a, c)$ *(transitivité)*
> - $\forall a, b \in O, S(a, b) \leq \infty$ *(finitude)*

La dissimilarité est parfois utilisée au lieu de la similarité. Elle est définie de manière analogue à la similarité, sauf qu'elle n'est pas transitive :

> ***Définition 2*** *(Dissimilarité). La dissimilarité DS : O × O → R est une fonction d'une paire d'entités à un nombre réel exprimant la dissimilarité entre ces deux entités telle que:*
>
> - $\forall a, b \in O, DS(a, b) \geq 0$ *(positivité)*
> - $\forall a, b, c \in O, DS(a, a) \leq DS(b, c)$ *et* $DS(a, a) = 0$ *(minimalité)*
> - $\forall a, b \in O, DS(a, b) = DS(b, a)$ *(symétrie)*
> - $\forall a, b \in O, DS(a, b) \leq \infty$ *(finitude)*

La distance est une mesure utilisée aussi souvent que les mesures de similarité. Elle mesure la dissimilarité de deux entités, elle est inverse de la similarité : si la valeur de la fonction de similarité de deux entités est élevée, la distance entre elles est petite et vice-versa. Elle est donc définie dans [Euzenat *et al*., 2004] comme suit :

> ***Définition 3*** *(Distance). La distance D : O × O → R est une fonction de la dissimilarité satisfaisant la définitivité et l'inégalité triangulaire :*
>
> - $\forall a, b \in O, D(a, b) = 0 \Leftrightarrow a = b$ *(définitivité)*
> - $\forall a, b, c \in O, D(a, b) + D(b, c) \geq D(a, c)$ *(inégalité triangulaire)*

Les valeurs de similarité sont souvent normalisées pour pouvoir être combinées dans des formules plus complexes. Si la valeur de similarité et la valeur de dissimilarité entre deux entités sont normalisées, notées $\overline{S}$ et $\overline{DS}$, alors on a $\overline{S} + \overline{DS} = 1$.

> ***Définition 4*** *(Normalisation). Une mesure est une mesure normalisée si les valeurs calculées par cette mesure ne peuvent varier que dans un intervalle de 0 à 1. Ces valeurs calculées sont appelées* valeurs normalisées*. Les fonctions du calcul sont appelées* fonctions normalisées *et notées* $\overline{f}$.

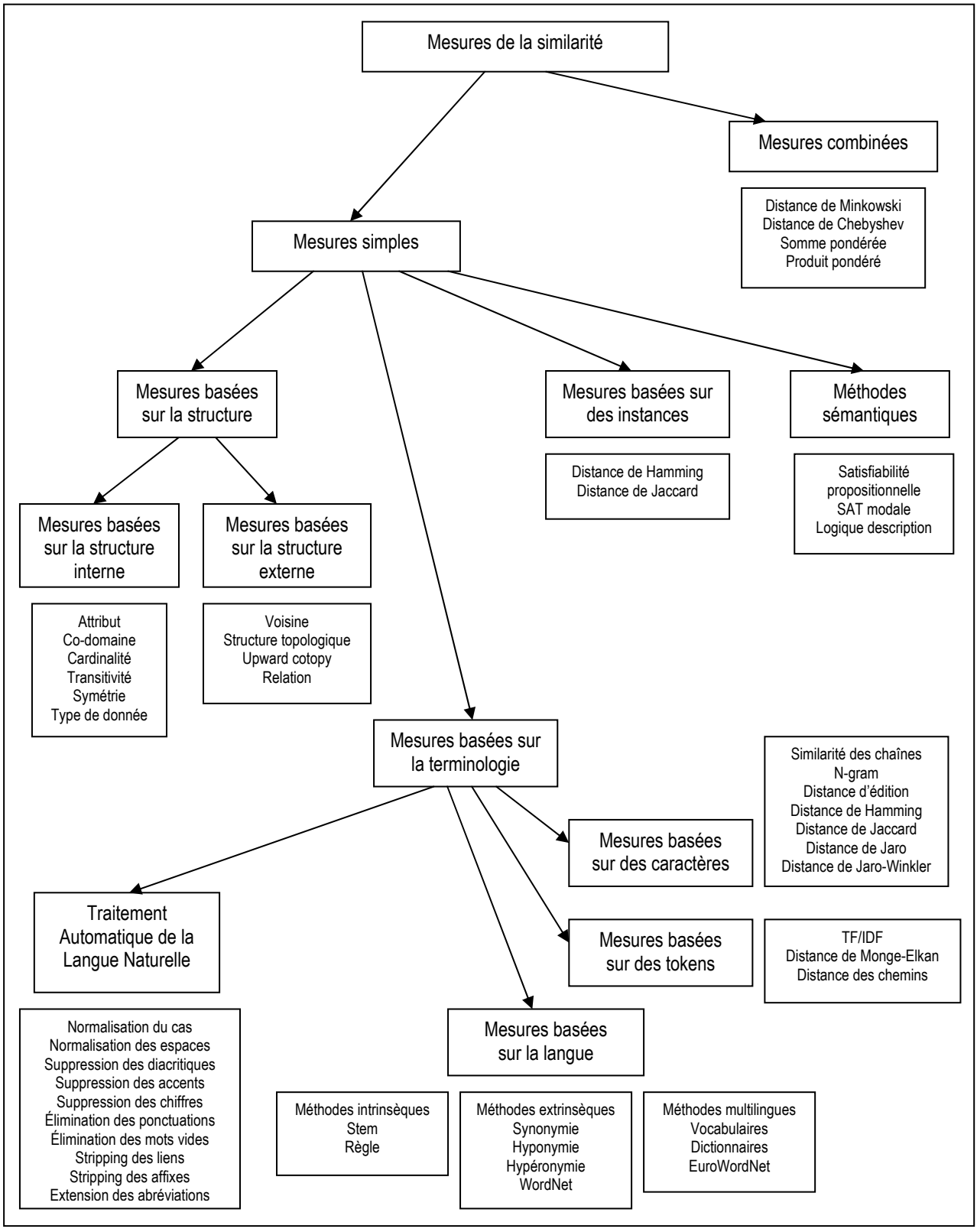


Figure 5 Les catégories des mesures de similarité selon différentes techniques

Les mesures de la similarité, de la dissimilarité, de la distance peuvent être classées selon la nature des entités que l'on veut comparer : des termes, des chaînes de caractères, des structures, des instances (des individus des classes), des modèles théoriques.

La *Figure 5* résume différentes mesures de la similarité, catégorisées selon les techniques utilisées. Ce résumé est une synthèse des travaux présentés dans [Rahm et Bernstein, 2001], [Euzenat *et al.*, 2004] et [Shvaiko et Euzenat, 2004].

1.2.1.2 Les méthodes terminologiques

Ces méthodes se basent sur la comparaison des termes ou des chaînes de caractères ou bien les textes. Elles sont employées pour calculer la valeur de la similarité des entités textuelles, telles que des noms, des étiquettes, des commentaires, des descriptions… Ces méthodes peuvent encore être divisées en deux sous-catégories : l'une contient des méthodes qui comparent des termes en se basant sur les caractères contenus dans ces termes et l'autre utilise certaines connaissances linguistiques.

1.2.1.2.1 Les méthodes se basent sur des chaînes de caractères

Ces méthodes analysent la structure des chaînes de caractères, l'ordre des caractères dans la chaîne, le nombre d'apparitions d'une lettre dans une chaîne pour concevoir des mesures de la similarité. Par contre, elles n'exploitent pas la signification des termes. Par exemple, les mesures dans cette catégorie retournent une grande valeur de similarité (jusqu'à 1) si elles comparent les termes « Voiture » et « voitures », mais une petite valeur, voire la valeur 0, si elles comparent les termes « voiture » et « bagnole ».

Les résultats de la comparaison des chaînes de caractères seront améliorés si ces chaînes sont « nettoyées » ou traitées avant de les fournir aux formules calculant la similarité. Cette phase est appelée la phase de normalisation ou de normalisation textuelle, qui diffère de la normalisation des valeurs de similarité dans un intervalle de [0, 1] discutée ci-dessus (*Définition 4*). Les différents types de normalisation textuelle sont ceux empruntés au domaine de traitement automatique de la langue naturelle (TALN) :

- *Normalisation des caractères* : ce type de normalisation convertit toutes les majuscules dans une chaîne de caractères en leurs formes minuscules ou vice-versa. Par exemple, la chaîne de caractères « *VoitureS* » sera convertie à « *voitures* » et ensuite, elle est considérée comme égale exactement à l'autre chaîne de caractères « *voitures* ».

- *Normalisation des espaces* : ce type de normalisation remplace toutes les séquences consécutives des espaces, des tabulations, des retours de chariot (les caractères CR) trouvées dans une chaîne de caractères par un seul caractère d'espace. Par exemple, l'expression « *ma voiture* » est normalisée à « *ma voiture* ».

- *Suppression des signes diacritiques ou des accents* (aigus, graves…) : ce type de traitement remplace des caractères avec des signes diacritiques par caractères correspondants sans signes diacritiques. Par exemple, le mot

« *Hanoï* » est remplacé par le mot « *Hanoi* » sans changer la signification du mot, la capitale du Vietnam. Cependant, certaines suppressions changeront la signification du terme : « *là* » (adverbe de lieu) et « *la* » (article).

- Suppression des chiffres.
- Élimination des ponctuations.
- Élimination des mots vides (les mots contenant peu d'informations tels que « est », « un », « les »…)
- Suppression des affixes (préfixes, suffixes).
- Extension des abréviations.
- Tokenisation.
- Lemmatisation (passer au singulier, à l'infinitif pour les verbes, au masculin pour les adjectifs…)

Il existe plusieurs mesures calculant la valeur de similarité ou la distance entre deux chaînes de caractères dans la littérature telles que la similarité de Jaccard, la distance de Hamming, la distance de Levenshtein… Certaines sont implémentées en Java[1], en C[2]… Nous présentons ici quelques mesures les plus utilisées dans les approches d'alignement d'ontologies dans le cadre du Web sémantique.

Si nous considérons une chaîne de caractères s comme un ensemble de caractères S, la similarité de Jaccard entre deux chaînes est définie ainsi :

Définition 5 *(Similarité de Jaccard). Soit s et t deux chaînes de caractères. Soit S et T les ensembles des caractères de s et t respectivement. La similarité de Jaccard est une fonction de la similarité* $S_{Jaccard}$ *:* $\mathbb{S} \times \mathbb{S} \rightarrow$ *[0, 1] telle que :*

$$\overline{S_{\text{Jaccard}}}(s,t) = \frac{|S \cap T|}{|S \cup T|}$$

Une classe importante des fonctions mesurant la distance entre deux chaînes de caractères `s` et `t`, est constituée des fonctions calculant la *distance d'édition* (edit distance), dans lesquelles la distance est le coût de la meilleure séquence des *opérations d'édition* qui convertissent `s` en `t`. Des opérations d'édition typiques sont celles de l'insertion, de la suppression, et de la substitution de caractère, et à chaque opération est affecté à un coût.

[1] http://www.dcs.shef.ac.uk/~sam/stringmetrics.html

[2] http://www.monkey.org/~jose/software/libdistance/distance_3.html

Définition 6 *(Distance d'édition). Soit un ensemble d'opérations d'édition OP, op ∈ OP, op : $\mathcal{S} \rightarrow \mathcal{S}$, et une fonction de coût d'édition w : OP → $\mathcal{R}$. Pour n'importe quelle paire des chaînes de caractères, il existe une séquence des opérations d'édition qui transforme la première en la seconde (et vice versa), la distance d'édition de deux chaînes de caractères s et t est une fonction de la dissimilarité DS_{de} : $\mathcal{S} \times \mathcal{S} \rightarrow$ [0, 1] telle que $DS_{de}(s, t)$ est le coût de la séquence des opérations la moins coûteuse qui transforme s en t.*

$$\overline{DS_{de}}(s,t) = \min_{(op_i)_i : op_n(\ldots op_1(s))=t} \left(\sum_{i \in I} w_{op_i} \right)$$

Les variantes de cette mesure sont différentes au niveau des coûts assignés aux opérations d'édition. On peut trouver :

- La distance de Levenstein où tous les coûts sont égaux à 1.
- La distance de Damerau qui est presque identique à la distance de Levenshtein mais qui peut tolérer des caractères adjacents qui ont été permutés, une erreur typographique habituelle.
- La distance de Smith-Waterman [Durban *et al.* 1998] qui affecte des coûts relativement inférieurs à la séquence des insertions ou des suppressions.
- La distance de Needleman-Wunsch avec des matrices des coûts pour chaque paire des caractères dans des opérations des insertions ou des suppressions.

La métrique Jaro [Jaro, 1995; 1989] produit la similarité entre deux chaînes de caractères en se basant sur le nombre et l'ordre des caractères communs entre elles.

Définition 7 *(Distance de Jaro). Soit s et t deux chaînes de caractères. Soit N_c le nombre des caractères communs apparaissant dans les deux chaînes dans une distance de moitié de la longueur de la chaîne la plus courte. Soit N_t le nombre des caractères transposés, qui sont des caractères communs apparaissant dans des positions différentes. La distance de Jaro est une fonction de la dissimilarité DS_{Jaro} : $\mathcal{S} \times \mathcal{S} \rightarrow$ [0, 1] telle que :*

$$\overline{DS_{\text{Jaro}}}(s,t) = 1 - \frac{1}{3}\left(\frac{N_c}{|s|} + \frac{N_c}{|t|} + \frac{N_c - N_t/2}{N_c} \right)$$

Il existe aussi des distances qui sont des variantes de la distance de Jaro, telles que la distance Jaro-Winkler [Winkler, 1999] :

Définition 8 *(Distance de Jaro-Winkler). Soit s et t deux chaînes de caractères. Soit P la longueur du préfixe commun le plus long de s et t. Soit n un nombre positif. La distance de Jaro-Winkler est une fonction de la dissimilarité $DS_{JaroWinkler}$: $\mathcal{S} \times \mathcal{S} \rightarrow$ [0, 1] telle que :*

$$\overline{DS_{\text{JaroWinkler}}}(s,t) = \overline{DS_{Jaro}}(s,t) - \frac{\max(P,n)}{10}\overline{DS_{Jaro}}(s,t)$$

1.2.1.2.2 Les distances basées sur des tokens

Les mesures présentées ci-dessus s'adaptent bien lorsque l'on veut comparer deux termes ou deux courtes chaînes de caractères. Il existe aussi des cas où l'on a besoin de comparer des textes longs ou bien des documents textuels. Dans ces cas, ces entités sont découpées en plusieurs morceaux, appelés *tokens*. Elles deviennent des ensembles des tokens, et la similarité entre elles est produite grâce aux mesures de similarité basées sur des tokens.

Il existe plusieurs mesures de cette catégorie dans la littérature telles que la similarité de Dagan [Dagan *et al.*, 1999], la distance de Jensen-Shannon, la distance de Fellegi-Sunter [Fellegi *et* Sunter, 1969]… Nous présentons ici quelques mesures les plus utilisées dans les approches d'alignement d'ontologies dans le cadre du Web sémantique.

La similarité de Jaccard (*Définition 5*) peut être étendue pour comparer des ensembles des tokens, en définissant la similarité comme le rapport entre la cardinalité de l'intersection des ensembles sur la cardinalité de leur union.

Une mesure qui est largement employée dans le domaine de la recherche d'information, semble convenable ici. Il s'agit du TF/IDF (Term frequency/Inverse Document Frequency). Dans sa conception originale, TF/IDF est employé pour mesurer la pertinence d'un terme dans l'ensemble de documents. La fréquence de terme, TF, dans un document donné montre l'importance de ce terme dans le document en question. La fréquence inverse de document, IDF, est une mesure de l'importance générale du terme dans l'ensemble de documents.

> ***Définition 9*** *(TF/IDF). Soit D un corpus des documents, |D| dénote le nombre des documents dans le corpus D. Soit t un terme à considérer, n(t) étant le nombre d'occurrences du terme t dans un document, et N étant le nombre des termes dans ce document, et d(t) étant le nombre des documents qui contiennent au moins une fois le terme t. Les mesures de TF et TF/IDF sont définies comme suivante :*
>
> $$TF = \frac{n(t)}{N} \text{ et } TF/IDF = TF * \log\left(\frac{|D|}{d(t)}\right)$$

La similarité des entités textuelles peut être construite comme la valeur cosinus de deux vecteurs représentant ces entités, où chaque dimension correspond à un terme et sa valeur correspond à la valeur TF/IDF de ce terme.

[Monge et Elkan, 1996] propose une méthode hybride pour comparer des chaînes de caractères longues, qui découpe ces deux chaînes en plusieurs chaînes plus courtes. Ensuite, ces dernières sont comparées par une mesure de distance quelconque citée ci-dessus. Enfin, les résultats obtenus sont combinés.

Définition 10 *(Similarité hybride). Soit $s = a_1...a_K$ et $t = b_1...b_L$ deux chaînes de caractères, où a_i et b_j sont des sous-chaînes de s et t respectivement. Soit S une mesure de la similarité entre deux chaînes des caractères. La similarité hybride est une fonction de la similarité $S_{Hybride}$: $\mathcal{S} \times \mathcal{S} \rightarrow [0, 1]$ telle que :*

$$\overline{S_{\text{Hybride}}}(s,t) = \frac{1}{K}\sum_{i=1}^{K} \max_{j=1}^{L} S(a_i, b_j)$$

[Cohen *et al.*, 2003] fait une excellente comparaison de plusieurs mesures de similarité montrant le point fort de chaque technique pour une tâche particulière. Chaque mesure de distance ou de similarité s'adapte mieux dans certains domaines d'application. Le *Tableau 1* (synthèse de [Cohen *et al.*, 2003] et [Scannapieco, 2004]) résume des domaines d'applications pour des mesures présentées dans cette partie.

Mesure de similarité	Domaine d'application
N-gram	Bigrams (n = 2) est efficace avec des erreurs typographiques mineures
Distance d'édition	Peut être appliquée aux entités ayant une longueur variable. Pour atteindre une exactitude raisonnable, les coûts des opérations de modification dépendent de chaque domaine
Distance de Hamming	Utilisée principalement pour les entités numériques ayant des tailles fixes, comme les codes postaux ou les numéros de sécurité sociale
Distance de Monge-Elkan	La meilleure performance au niveau des résultats dans plusieurs expériences. Peut être employée dans plusieurs domaines
Distance de Jaro/Jaro-Winkler	Presque même performance au niveau des résultats que Monge-Elkan mais beaucoup plus rapide
Distance basée sur TF/IDF	La meilleure pour la comparaison des textes longs (basée sur des tokens)

Tableau 1 Critères principaux d'utilisation des mesures de la similarité

1.2.1.2.3 Les méthodes linguistiques

La similarité entre deux entités représentées par des termes peut aussi être déduite en analysant ces termes à l'aide des méthodes linguistiques. Ces méthodes exploitent essentiellement des propriétés expressives et productives de la langue naturelle [Maynard et Ananiadou, 1999]. Les informations exploitées peuvent être celles intrinsèques (des propriétés linguistiques internes des termes telles que des propriétés morphologiques ou syntaxiques) ou celles extrinsèques (employant des ressources externes telles que des vocabulaires ou des dictionnaires).

Les méthodes intrinsèques

Une même entité ou un même concept peut être référencé par plusieurs termes (synonymie) ou par plusieurs variantes d'un même terme. Le *Tableau 2* (extrait de [Euzenat *et al.*, 2004]) montre des variantes possibles du termes *enzyme activity*.

Type	Sous-type	Exemple
Morphologique	Inflexion	enzyme activities
	Dérivation	enzymatic activity
	Flexionnel-Dérivationnel	enzymatic activities
Syntaxique	Insertion	enzyme amidolytic activity
	Permutation	activity of enzyme
	Coordination	enzyme and bactericidal activity
Morpho-syntaxiques	Dérivation-Coordination	enzymatic and bactericidal activity
	Inflexion- Permutation	activity of enzymes
Sémantique		fermentation
Multilingue	French	activité d'enzyme
	Vietnamien	sự lên men

Tableau 2 Variantes du terme *enzyme activity* (extrait de [Euzenat *et al.*, 2004])

Les méthodes intrinsèques fonctionnent avec le principe de chercher la forme canonique ou représentative d'un mot ou d'un terme (lemme) à partir de ses variantes linguistiques (lexème). La similarité entre deux termes est donc décidée en comparant leurs lemmes. Par exemple, le résultat de la mesure de similarité exacte de deux mots « ran » et « running » sera égal à 0 (c-a-d. ils sont différents), alors que le résultat de la même mesure pour les lemmes de ces mots sera égal à 1, ce qui indique que « ran » et « running » sont similaires.

La recherche du lemme d'un mot peut être effectuée dans un dictionnaire. Une autre approche qui est automatique et plus légère et plus efficace est d'utiliser des *stemmers*. Un stemmer est un programme ou un algorithme qui détermine la forme radicale à partir d'une forme infléchie ou dérivée d'un mot donné. Les radicaux (stems) trouvés par les stemmers n'ont pas besoin d'être identiques à la racine morphologique du mot. Il suffit que les mots similaires soient associés à un même radical, même si ce radical n'est pas une racine de mot valide. Un stemmer pour le français, par exemple, devrait identifier les chaînes de caractères « maintenaient », « maintenait », « maintenant », ou « maintenir » comme basées sur la racine "mainten".

Une approche plus complexe pour déterminer le radical exact d'un mot est la *lemmatisation*. Ce processus comprend la détermination de la partie du discours (catégorie lexicologique) d'un mot, et l'application des règles de normalisation différentes pour chaque partie du discours. Cette approche exige la connaissance de la grammaire d'une langue, des règles différentes… Elle est donc lourde, compliquée et difficile à implémenter.

Le premier stemmer publié a été écrit par Julie Beth Lovins [Lovins, 1968]. Ensuite un autre stemmer a été développé par Martin Porter [Porter, 1980]. Ce dernier est très largement utilisé, et est devenu l'algorithme standard utilisé pour chercher des radicaux dans la langue anglaise. L'algorithme est implémenté dans plusieurs langages de

programmation. Snowball[1], un framework pour implémenter des algorithmes de stemmers, leurs variantes ou bien des algorithmes de stemmers pour des autres langues (par exemple le finnois, le russe, le danois l'allemand, le français…), est aussi créé par Porter.

Les méthodes extrinsèques

Ces méthodes calculent la valeur de similarité entre deux termes en employant des ressources externes telles que des dictionnaires, des lexiques ou des vocabulaires. La similarité est décidée grâce aux liens sémantiques déjà existants dans ces ressources externes tels que des liens synonymes (pour l'équivalence), des liens hyponymes/ hypernymes (pour la subsomption). Par exemple, à l'aide des ressources des synonymes, « voiture » et « bagnole » sont dites similaires. Typiquement, WordNet[2], un système lexicologique, est employé pour trouver des relations telles que la synonymie entre des termes, ou pour calculer la distance sémantique entre ces termes, en utilisant des liens sémantiques dans WordNet, afin de décider s'il existe une relation entre eux.

Les ressources externes utilisées dans les méthodes extrinsèques peuvent aussi être des vocabulaires ou des dictionnaires multilingues, ou d'autres systèmes tels que EuroWordNet[3], Polylex[4].

1.2.1.3 Les méthodes structurelles

Ce sont des méthodes qui déduisent la similarité de deux entités en exploitant des informations structurelles lorsque les entités en question sont reliées aux autres par des liens sémantiques ou syntaxiques, formant ainsi une hiérarchie ou un graphe des entités. Nous appelons *méthodes structurelles internes* les méthodes qui n'exploitent que des informations concernant des attributs d'entité, et *méthodes structurelles externes* les autres qui considèrent des relations entre des entités.

1.2.1.3.1 Les méthodes structurelles internes

Ces méthodes calculent la similarité entre deux entités en exploitant des informations des structures internes de ces entités. Dans la plupart des cas, ce sont des informations concernant des attributs de l'entité, telles que des informations du co-domaine des attributs, celles de la cardinalité des attributs, celles des caractéristiques des

[1] http://snowball.tartarus.org/

[2] http://wordnet.princeton.edu/

[3] http://www.illc.uva.nl/EuroWordNet/ - un système de réseaux sémantiques pour des langues européennes où chaque langue développe son propre WordNet et elles sont reliées entre elles par des liens inter-langues

[4] http://www.informatics.susx.ac.uk/research/nlp/polylex/ - un lexique multilingue pour le Néerlandais, l'Anglais et l'Allemand, construit à partir de différents lexiques monolingues contenues dans la base de données CELEX (http://www.ru.nl/celex/)

attributs (la transitivité, la symétrie), ou celles des autres types de restriction sur des attributs. Par exemple, en considérant l'entité : le concept « Humain », nous pouvons exploiter des informations concernant des attributs de ce concept tels que l'intervalle des valeurs de donnée pour l'attribut « hasAge », à savoir [0, 150] ; la cardinalité de l'attribut « hasSpouse », à savoir 1 ; ou bien la caractéristique transitive de l'attribut « hasAncestor ».

Dans le domaine des bases de données, plusieurs méthodes ont été proposées pour calculer la similarité entre deux éléments de deux schémas de base de données, en se basant sur les contraintes à propos de ces éléments. Dans la revue [Rahm et Bernstein, 2001], nous pouvons trouver l'algorithme Cupid [Madhavan *et al.*, 2001] qui cherche des correspondances entre des éléments en se basant entre autres, sur la compatibilité des attributs, des types de données ; l'approche SEMINT [Li et Clifton, 2000] qui identifie des correspondances des attributs en se basant sur des informations de schéma (telles que des types de donnée, la longueur, la précision, l'existence des clés, des contraintes de co-domaine ou de valeur, l'autorisation des valeurs nulles…) et sur les statistiques des instances (tel que le maximum, le minimum, la moyenne, le coefficient de variance, l'existence des valeurs nulles ou des décimales, le groupe, ou le nombre des segments).

[Valtchev, 1999] propose une mesure de la similarité entre deux types de données. Cette mesure se base sur la différence entre deux tailles des types (la taille d'un type est définie comme la cardinalité de l'ensemble de valeurs qu'il définit) et sur la taille de leur généralisation commune (dont la définition dépend du type : il s'agit d'un ensemble pour les types énumérés, d'un intervalle pour les types ordonnés…).

1.2.1.3.2 Les méthodes structurelles externes

Contrairement aux méthodes décrites dans *1.2.1.3.1*, qui exploitent des informations des attributs d'entité, les méthodes structurelles externes exploitent des relations entre des entités elles-mêmes, qui sont souvent des relations de subsomption (is-a ou spécialisation) ou de méréologie (part-whole). Avec ces relations, les entités sont considérées dans des hiérarchies et la similarité entre elles est déduite de l'analyse de leurs positions dans ces hiérarchies. L'idée de base est que si deux entités sont similaires, leurs voisines pourraient également être d'une façon ou d'une autre similaires. Cette observation peut être exploitée de plusieurs manières différentes en regardant des relations avec d'autres entités dans des hiérarchies. Deux entités peuvent être considérées similaires si :

- Leurs super-entités directes (ou toutes leurs super-entités) sont similaires.
- Leurs sœurs (ou toutes leurs sœurs, qui sont les entités ayant la même super-entité directe avec les entités en question) sont déjà similaires.
- Leurs sous-entités directes (ou toutes leurs sous-entités) sont déjà similaires.
- Leurs descendants (entités dans le sous-arbre ayant pour racine l'entité en question) sont déjà similaires.

- Toutes (ou presque toutes) leurs feuilles (les entités de même type, qui n'ont aucune sous-entité, dans le sous-arbre ayant pour racine l'entité en question) sont déjà similaires.

- Toutes (ou presque toutes) les entités dans les chemins de la racine aux entités en question sont déjà similaires.

Des combinaisons des heuristiques ci-dessus sont aussi possibles.

Cependant, cette approche peut rencontrer quelques difficultés dans les cas, où les hiérarchies sont différentes au niveau de granularité. Par exemple, si dans une hiérarchie, l'entité « Personne » a deux sous-entités « Enfant » et « Adulte », et si dans une autre hiérarchie, la même entité « Personne » est divisée en deux autres sous-entités « Femme » et « Homme », la déduction que « Enfant » et « Femme » ou « Enfant » et « Homme » sont similaires, est incorrecte dans tous les cas.

Des mesures ont été proposées pour comparer deux entités basées sur des structures taxonomiques. [Valtchev et Euzenat, 1997] proposent une mesure récursive de la *dissimilarité topologique structurelle* qui se base sur la distance du chemin le plus court dans un graphe. [Mädche et Staab, 2002] introduit la notion de « upward cotopy » qui est définie UC(c, H) = {c'∈H ; c ≤ c'}, l'ensemble d'entités qui sont super-entités de l'entité c dans la hiérarchie H. La mesure de similarité, inspirée de la distance de Jaccard (voir *Définition 5*), est alors définie ainsi :

> ***Définition 11*** *(Similarité de « upward cotopy »). Soit H_1 et H_2 deux hiérarchies. Soit e et e' deux entités dans H_1 et H_2 respectivement. La similarité de « upward cotopy » entre e et e' est une fonction de la similarité S_{cotopy} : O x O → [0, 1] telle que :*
>
> $$\overline{S_{cotopy}}(e,e') = \frac{|UC(e,H_1) \cap UC(e',H_2)|}{|UC(e,H_1) \cup UC(e',H_2)|}$$

Une autre approche pour déduire la similarité entre deux entités exploite des relations reliant ces deux entités. L'idée est que si l'on a deux entités similaires A et A', et si elles sont connectées par un même type de relation R avec deux autres entités B et B', alors on peut déduire que B et B' sont d'une façon ou d'une autre similaires. De même, si on sait que A et A' sont similaires, B et B' sont aussi similaires, alors les relations de A-B et de A'-B' peuvent être similaires. L'idée peut être étendue pour un ensemble d'entités et de relations : si on a un ensemble de relations $R_1 \ldots R_n$ qui sont similaires avec un autre ensemble de relations $R_1' \ldots R_n'$, alors les entités qui sont les domaines (ou les co-domaines) de ces relations sont considérées comme similaires.

Cependant, cette approche pose un autre problème : comment peut-on dire que deux relations sont similaires ? Cette approche est basée sur la similarité des relations pour impliquer la similarité de leurs entités de domaine ou leurs entités de co-domaine. Des relations entre les entités peuvent être considérées comme d'autres entités, elles peuvent être aussi organisées dans une hiérarchie de relations, et donc, le calcul de la similarité entre les relations est également un grand problème.

Une mesure de la similarité entre des relations définie d'après ce principe peut être trouvée dans [Mädche et Staab, 2002] :

Définition 12 *(Similarité des relations). Soit H_1 et H_2 deux hiérarchies. Soit r et r' deux relations dans H_1 et H_2 respectivement. Soit dom(r) et ran(r) deux fonctions qui retournent le domaine et le co-domaine de la relation r respectivement. La similarité des relations entre r et r' est une fonction de la similarité $S_{relation}$: O x O → [0, 1] telle que :*

$$\overline{S_{relation}}(r,r') = \sqrt{\overline{S_{cotopy}}(dom(r),dom(r')) * \overline{S_{cotopy}}(ran(r),ran(r'))}$$

1.2.1.4 Les méthodes extensionelles

Ces méthodes déduisent la similarité entre deux entités qui sont notamment des concepts ou des classes en analysant leurs extensions, c-a-d, leurs ensembles des instances.

Dans le cas où les ensembles des instances partagent une partie commune, on peut avoir des mesures qui emploient des opérations de l'ensemble, telles que celles de Hamming ou de Jaccard. Fondamentalement, la mesure de Hamming compte nombre d'éléments différents entre deux ensembles à comparer et la mesure de Jaccard est le rapport entre l'intersection des ensembles et leur union (voir *Définition 5*). Ces mesures peuvent être adaptées pour construire des mesures extensionnelles.

Définition 13 *(Distance de Hamming, version adaptée pour les ensembles des instances). Soit S et T deux ensembles. La distance de Hamming (appelée aussi la différence symétrique) entre S et T est une fonction de la dissimilarité $DS_{Hamming}$: 2^E x 2^E → [0, 1] telle que :*

$$\overline{DS_{\text{Hamming}}}(S,T) = \frac{|S \cup T - S \cap T|}{|S \cup T|}$$

Définition 14 *(Distance de Jaccard, version adaptée pour les ensembles des instances). Soit S et T deux ensembles. Soit P(x) la probabilité d'une instance aléatoire être dans l'ensemble X. La distance de Jaccard est une fonction de la dissimilarité $DS_{Jaccard}$: 2^E x 2^E → [0, 1] telle que :*

$$\overline{DS_{\text{Jaccard}}}(S,T) = 1 - \frac{P(S \cap T)}{P(S \cup T)}$$

Les mesures ci-dessus produisent la similarité de deux entités qui est en fait la similarité entre les deux ensembles de leurs instances en se basant sur la comparaison exacte des éléments dans deux ensembles. Dans le cas où les ensembles des instances ne partagent aucune partie commune, ces mesures ne sont plus applicables (le résultat retourné sera toujours égal à 1, c-a-d les entités à comparer sont toujours différentes).

Une autre mesure qui partage l'idée similaire avec la mesure hybride (*Définition 10*) est définie dans [Valtchev, 1999] comme la similarité basée sur des correspondances (match-based similarity). Ici, la similarité entre deux ensembles est la similarité moyenne des paires des éléments dans `Pairing`, où `Pairing` est l'ensemble de correspondances ayant la somme maximale des toutes les similarités des paires dans l'ensemble. Le calcul

de l'ensemble `Pairing(S, T)` est un problème d'optimisation qui maximise le total des similarités des paires des éléments de `S` et `T`.

> ***Définition 15*** *(Similarité basée sur des correspondances). Soit S et T deux ensembles d'entités. Soit $S_q(s, t)$ une mesure de similarité quelconque de deux entités s et t. Soit Pairing(S, T) l'ensemble de correspondances entre S et T ayant la somme maximale des valeurs de similarité de ces correspondances. La similarité basée sur des correspondances entre deux ensembles S et T est une fonction de la similarité $S_{Corr} : 2^E \times 2^E \rightarrow [0, 1]$ telle que :*

$$\overline{S_{\text{Corr}}}(S,T)=\frac{1}{max(|S|,|T|)}\sum_{(s,t)\in Pairing(S,T)} S_q(s,t)$$

Une autre mesure pour calculer la similarité entre deux ensembles emploie une technique d'analyse multidimensionnelle dans le domaine de la statistique [Cox, 1994]. L'hypothèse de cette technique est que si deux entités ont les distances très similaires à toutes autres entités, elles doivent être très similaires. Les entités dans des ensembles sont représentées par des vecteurs, dont la valeur d'une dimension est la similarité de l'entité en question avec une autre entité dans les deux ensembles. La similarité entre deux ensembles est donc la similarité (la valeur du cosinus) de deux vecteurs moyens de ces deux ensembles.

> ***Définition 16*** *(Similarité des ensembles). Soit $S = \{s_1, s_2, \ldots\}$ et $T = \{t_1, t_2, \ldots\}$ deux ensembles d'entités. Soit $S_q(s_i, t_j)$ une mesure de similarité quelconque de deux entités s et t. Soit $\vec{s_i} = (sim(e_i,e_1), sim(e_i,e_2), \ldots sim(e_i,f_1), sim(e_i,f_2), \ldots)$ le vecteur représentant de l'entité e_i. La similarité des ensembles entre S et T est une fonction de la similarité $S_{Set} : 2^E \times 2^E \rightarrow [0, 1]$ telle que :*

$$\overline{S_{\text{Set}}}(S,T)=\frac{\sum_{s\in S}\vec{s}}{\left|\sum_{s\in S}\vec{s}\right|}\otimes\frac{\sum_{t\in T}\vec{t}}{\left|\sum_{t\in T}\vec{t}\right|}$$

1.2.1.5 Les méthodes sémantiques

Les méthodes sémantiques se basent sur des modèles de logique (tels que la satisfiabilité propositionnelle (SAT), la SAT modale ou les logiques de descriptions) et sur des méthodes de déduction pour déduire la similarité entre deux entités.

Les approches dans [Giunchiglia et Shvaiko, 2004; Giunchiglia *et al.*, 2004; Bouquet *et al.*, 2003] emploient des techniques de satisfiabilité propositionnelle (SAT) pour vérifier la validité d'un ensemble de formules propositionnelles qui est construit en traduisant des relations déjà connues et des relations à vérifier entre des entités vers des formules propositionnelles.

L'approche dans [Shvaiko, 2004] étend les méthodes proposées ci-dessus, qui sont pour le modèle de la SAT propositionnelles, vers le modèle de la SAT modale, qui peut aussi contenir des prédicats binaires. La limite du premier modèle est qu'il n'accepte que des prédicats unaires qui sont des entités comme des concepts, des classes. Le dernier permet de calculer en plus avec des prédicats binaires tels que des relations, des attributs

ou des propriétés (slots) et d'employer des opérateurs de la logique modale. La validité de l'ensemble de formules formulées en logique modale est aussi vérifiée en utilisant des procédures de recherche de la satisfiabilité (SAT). Si la validité est satisfaite, les relations hypothétiques entre des entités, qui sont des traductions de la requête sur la relation entre ces entités en logique modale, sont confirmées.

Les techniques des logiques de description (telles que le test de subsomption) peuvent être employées pour vérifier des relations sémantiques entre des entités telles que l'équivalence (la similarité est égale à 1), la subsomption (la similarité est de 0 à 1) ou l'exclusion (la similarité est égale à 0), et permettent donc de déduire la similarité de deux entités.

1.2.2 Les méthodes de combinaison des similarités

Une entité peut être considérée sous plusieurs différents aspects, soit en s'appuyant sur son nom, soit sur ses attributs, ou soit sur ses relations avec d'autres entités. La similarité entre deux entités peut donc être calculée en se basant sur plusieurs aspects. Sur chaque aspect, les caractéristiques d'une entité sont comparées avec les caractéristiques correspondantes d'une autre par une des mesures de similarité de base présentées dans la section *1.2.1*, cela retourne une valeur de la similarité (ou de la dissimilarité/distance). Il faut donc un moyen pour combiner toutes les valeurs de similarité calculées de chaque aspect pour produire une seule valeur de similarité représentative pour deux entités à comparer. Cette partie analyse quelques approches existantes dans la littérature.

La distance de Minkowski entre deux entités est définie comme suivante :

Définition 17 *(Distance de Minkowski). Soit O l'ensemble d'objets qui peuvent être analysés dans n dimensions. Soit x et y deux objets dans O. La distance de Minkowski entre x et y est une fonction de la dissimilarité* $DS_{Minkowski} : O \times O \rightarrow \mathbb{R}$ *telle que :*

$$DS_{\text{Minkowski}}(x,y) = \sqrt[p]{\sum_{i=1}^{n} DS(x_i, y_i)^p}$$

Cette distance est une mesure généralisée avec différentes valeurs de p, $p \geq 1$. Quand p est égale à 1, elle devient la distance de « city block » et quand p = 2 elle devient la distance euclidienne. La distance de Chebyshev (appelée aussi la distance de valeur maximum) est un cas spécial de la distance de Minkowski avec $p = \infty$: $DS_{Chebyshev}(x,y) = \max_i DS(x_i, y_i)$

Cette mesure n'est une fonction linéaire que quand p = 1 ou $p = \infty$. Dans le cas où p = 1, une variante de cette mesure avec des poids est souvent utilisée. Le bon côté de cette variante est que nous pouvons contrôler l'influence (ou l'importance) de chaque dimension sur la valeur finale de la distance. Les dimensions plus importantes seront associées avec les poids plus élevés, donc les valeurs de ces dimensions influenceront mieux à la valeur agrégée finale.

Définition 18 *(Somme pondérée). Soit O l'ensemble d'objets qui peuvent être analysés dans n dimensions. Soit x et y deux objets dans O. Soit w_i le poids de la dimension i. Soit $DS(x_i, y_i)$ la dissimilarité de la paire des objets à la dimension i. La somme pondérée entre x et y est une fonction de la dissimilarité DS_{sp} : O x O → ℜ telle que :*

$$DS_{\text{sp}}(x,y)=\sum_{i=1}^{n} w_i * DS(x_i,y_i)$$

En général, la somme des poids est égale à 1 : $\sum_{i=1}^{n} w_i = 1$, dans ce cas, nous avons la version normalisée de DS_{sp}.

Une autre mesure analogue à la somme pondérée est le produit pondéré. Cependant, un inconvénient de cette mesure est que le résultat sera égal à 0 si une des dimensions est égale à 0.

Définition 19 *(Produit pondéré). Soit O l'ensemble d'objets qui peuvent être analysés dans n dimensions. Soit x et y deux objets dans O. Soit w_i le poids de la dimension i. Soit $DS(x_i, y_i)$ la dissimilarité de la paire des objets à la dimension i. Le produit pondéré entre x et y est une fonction de la dissimilarité DS_{pp} : O x O → ℜ telle que :*

$$DS_{\text{sp}}(x,y)=\prod_{i=1}^{n} DS(x_i,y_i)^{w_i}$$

Les approches d'alignement des schémas ou des ontologies présentées dans la section suivante emploient une ou plusieurs mesures de similarité présentées dans la section *1.2.1* pour calculer les valeurs de similarité entre des entités, ensuite retourner des alignements entre deux schémas ou deux ontologies en évaluant ces valeurs de similarité. Toutes les approches, qui combinent des valeurs de similarité calculées par différentes mesures, emploient la méthode de la somme pondérée (*Définition 18*). Cependant, certaines approches (par exemple Anchor-PROMPT) déduisent des alignements en examinant des critères heuristiques sans utiliser des méthodes de combinaison des similarités.

1.2.3 Les approches d'alignement d'ontologies

Dans cette partie, nous allons analyser des approches déjà existantes dans la littérature qui concernent le problème d'alignement d'ontologies. Comme nous l'avons souligné au début de la section *1.2*, les schémas, les répertoires, les vocabulaires peuvent, pour certains aspects, être considérés comme des ontologies simples, ne disposant que des concepts (ou classes) organisés ou non dans une hiérarchie de subsomption, sans autres relations entre ces concepts. Nous étudions donc quelques approches permettant d'établir des correspondances entre des schémas, des répertoires ou des vocabulaires dans des domaines tels que les bases de données, l'intégration des données…

Nous présentons tout d'abord une distinction entre les termes souvent utilisés : *fusion* (merge), *alignement* (align) et *intégration* (integration) :

La fusion est l'action de construire une nouvelle ontologie en unifiant plusieurs ontologies dans une seule ontologie [Pinto et Martins, 2001] [Stumme et Maedche, 2001]. Le but final est de créer une seule ontologie cohérente qui inclut toutes les informations de toutes les sources [Noy et Musen, 2000] [Klein, 2001].

L'alignement est employé quand des sources doivent être rendues conformes et cohérentes entre elles mais elles sont toujours gardées séparément [Noy et Musen, 2000]. Cela implique de mettre deux (ou plus) ontologies en accord mutuel, et de les rendre conformes et cohérentes [Corcho et Gomez-Perez, 2001] [Klein, 2001]. Plusieurs alignements sont créés pendant ce processus, pour définir collectivement les relations entre les ontologies originales.

L'intégration entraîne de construire une nouvelle ontologie en composant des pièces d'autres ontologies disponibles [Pinto et Martins, 2001]. Comme la fusion, ce processus produit comme résultat une nouvelle ontologie. La différence entre l'intégration et la fusion est que seules certaines parties des ontologies originales seront intégrées, le but n'est pas d'accomplir une fusion complète [Hameed *et al.*, 2003].

1.2.3.1 Les approches dans d'autres domaines que le Web sémantique

Rahm et Bernstein dans [Rahm et Bernstein, 2001] a fait une excellente revue des travaux attaquant le problème de mise en correspondance automatique des schémas dans le domaine de base de données. Cette revue a été citée dans beaucoup d'articles de recherche sur la mise en correspondance des schémas ou bien des ontologies. Elle inspire aussi d'autres revues telles que [Euzenat *et al.*, 2004], [Shvaiko et Euzenat, 2004]. Les approches revues sont SemInt [Li et Clifton, 2000], LSD [Doan *et al.*, 2001], SKAT [Mitra *et al.*, 1999], TranScm [Milo et Zohar, 1998], DIKE [Palopoli *et al.*, 1998], ARTEMIS [Castano *et al.*, 2001], CUPID [Madhavan *et al*., 2001].

Nous examinons un peu plus en détail certaines approches.

Similarity Flooding (SF) [Melnik *et al*., 2001] est un algorithme de mise en correspondance des graphes génériques. Son application à la mise en correspondance de schémas est présentée dans [Melnik *et al*., 2002]. SF convertit les schémas (SQL DDL, XML) en des graphes étiquetés orientés (directed labeled graphs) et puis il applique le calcul de point-fixe pour déterminer les nœuds similaires dans les graphes. Cette approche est basée sur une comparaison très simple des chaînes des caractères des noms des nœuds pour calculer des correspondances initiales, puis ces correspondances sont fournies au module de mise en correspondance structurel de SF. Bien que SF ait appliqué une nouvelle approche orientée structurelle basée sur l'intuition que les éléments de deux schémas distincts sont similaires quand leurs éléments adjacents sont déjà similaires pour propager la similitude de deux éléments à leurs voisins respectifs, il se fonde principalement sur des étiquettes des arcs dans les graphes. S'il n'y a aucune étiquette pour des arcs dans le graphe ou si ces étiquettes sont presque identiques, l'algorithme ne fonctionne pas bien. Sans utilisation d'un dictionnaire terminologique externe (tel que WordNet [Miller, 1995]), l'algorithme ne donne pas de bons résultats de mise en correspondance au niveau linguistique dans la première phase, qui seront fournis à la deuxième phase, influençant ainsi les résultats finaux.

Contrairement à SF, Cupid [Madhavan *et al.,* 2001] a proposé une approche de recherche des correspondances combinant un module sophistiqué de mise en correspondance des noms et un algorithme de mise en correspondance au niveau structurel. L'algorithme se compose de trois étapes : (i) le niveau linguistique : les valeurs de similarité entre les noms des éléments (les étiquettes) sont calculées en employant des techniques et mesures linguistiques (*1.2.1.2*) telles que la normalisation des chaînes des caractères (*1.2.1.2.1*), la catégorisation, les mesures de similarité sur des préfixes, des suffixes, l'emploi le thésaurus des synonymes, des hypernymes ; (ii) le niveau structurel : la similarité structurelle de deux éléments est la similarité de deux arbres dont leurs racines sont les éléments à comparer. Cette dernière similarité est calculée en se basant principalement sur la similarité des feuilles de ces arbres. (iii) la valeur de similarité finale de deux éléments est la somme pondérée de deux valeurs calculées dans deux étapes précédentes. Si cette valeur finale est plus grande qu'un seuil prédéfini, deux éléments sont considérés similaires. Dans son approche, Cupid produit la valeur de similarité de deux éléments en se basant principalement sur la similitude des éléments atomiques dans les graphes (c'est-à-dire des feuilles). Ainsi s'il y a des différences significatives en structure des graphes donnés, Cupid ne peut pas trouver des correspondances correctes. Par exemple, si un concept est situé à la place d'une feuille dans le premier graphe (schéma), mais dans le deuxième, il est à la place d'élément non-feuille qui est la racine d'un sous-graphe, Cupid ne détectera pas qu'il s'agit du même concept.

Do et Rahm ont développé COMA [Do et Rahm, 2002] comme un système permettant de mettre en correspondance des schémas (des bases des données, de XML) automatiquement ou bien manuellement. Le système fournit une bibliothèque des algorithmes de mise en correspondance de base (appelés matchers) et quelques mécanismes pour combiner des résultats des algorithmes de base afin d'obtenir une valeur de similarité finale de deux éléments dans deux schémas. La bibliothèque des matchers se compose de (i) 6 matchers simples qui emploient des techniques linguistiques (*1.2.1.2*) telles que la similarité des préfixes, des suffixes, n-gram, la distance d'édition (*Définition 6*), la similarité phonétique (soundex), la synonymie avec des dictionnaires externes, la similarité des types de données prédéfinie ; (ii) 5 matchers hybrides qui combinent des matchers simples précédents en exploitant quelques informations structurelles telles que des chemins entre des éléments, la similarité de leurs enfants. COMA présente quelques stratégies pour agréger des résultats de similarité pour chaque paire d'éléments, calculés de différents matchers tels que le choix de la valeur maximale, la valeur moyenne, la somme pondérée ou la valeur minimale. La sélection de bonnes correspondances parmi toutes les paires d'éléments repose sur une des stratégies suivantes : la paire ayant la valeur de similarité agrégée maximale, ou les paires ayant les valeurs de similarité agrégées dépassant un seuil. Le système COMA permet aussi d'exécuter plusieurs itérations de calcul et d'utiliser le résultat de l'étape précédente dans le calcul de similarité de l'étape actuelle, ou d'avoir des interactions des utilisateurs dans chaque itération.

1.2.3.2 Les approches dans le domaine du Web sémantique

Dans [Dieng et Hug, 1998a], les auteurs considèrent un contexte où les experts peuvent employer leurs propres ontologies, appelées les *ontologies personnelles*. Une ontologie est alors représentée par un support dans le formalisme des graphes

conceptuels : ce support comprend un treillis de types de concepts, une hiérarchie de types de relations et un ensemble de marqueurs permettant de désigner les instances. L'objectif est de construire un modèle de connaissances commun (ontologie commune) à partir de différents modèles de connaissances des experts (ontologie personnelle). Cela est réalisé, dans un système appelé MULTIKAT, par la comparaison des ontologies personnelles en exploitant des techniques reposant sur des opérations du formalisme de graphe conceptuel ou sur la structure des graphes. La comparaison de deux supports est effectuée en trois étapes : la comparaison et la fusion des treillis de type de concept, la comparaison et la fusion des hiérarchies de type de relation et la comparaison et la fusion de deux ensembles de marqueurs. Les techniques utilisées pour obtenir la similarité entre deux entités sont alors : l'égalité des chaînes de caractères sur des noms de type, des synonymes, la comparaison des sous-types et des super-types directs dans la hiérarchie de types (*1.2.1.3.2*), la somme pondérée (*Définition 18*) pour combiner des valeurs de similarité.

Anchor-PROMPT [Noy et Musen, 2001] construit un graphe étiqueté orienté représentant l'ontologie à partir de la hiérarchie des concepts (appelés classes dans l'algorithme) et de la hiérarchie des relations (appelées slots dans l'algorithme), où les noeuds dans le graphe sont des concepts et les arcs dénotent des relations entre les concepts (les étiquettes des arcs sont les noms des relations). Une liste initiale des paires d'ancres (des paires de concepts similaires) définies par les utilisateurs ou automatiquement identifiées par la mise en correspondance lexicologique sert d'entrée à l'algorithme. Anchor-PROMPT analyse alors les chemins dans les sous-graphes limités par les ancres et il détermine quels concepts apparaissent fréquemment en positions similaires sur les chemins similaires. En s'appuyant sur ces fréquences, l'algorithme décide si ces concepts sont sémantiquement similaires. Cependant, Anchor-PROMPT ne cherche que des correspondances des concepts, pas des correspondances des relations. En outre, il emploie des noms de relation pour des étiquettes sur les arcs et la comparaison des chaînes de caractères de ces étiquettes n'est que la comparaison simple. Ainsi si les noms de relation sont différemment définis, l'algorithme ne fonctionnera pas bien. Les résultats retournés par l'algorithme seront également limités si les structures des ontologies sont différentes (par exemple l'une est profonde avec beaucoup de concepts au milieu, et l'autre est peu profonde). L'algorithme rencontre des problèmes si une hiérarchie a seulement quelques niveaux et si la plupart des relations sont associées aux concepts au-dessus de la hiérarchie.

GLUE [Doan *et al.,* 2002] est la version évoluée de LSD [Doan *et al.,* 2000] dont le but est de trouver semi-automatiquement des correspondances entre des schémas pour l'intégration de données. Comme LSD, GLUE utilise la technique d'apprentissage (telle que Naive Bayes) pour trouver des correspondances entre deux ontologies. GLUE comprend plusieurs modules d'apprentissage (learners), qui sont entraînés par des instances des ontologies. Ces modules emploient la technique d'apprentissage Bayes naïf [Domingos et Pazzani, 1997] en exploitant différentes caractéristiques des instances telles que les valeurs textuelles des instances, les noms des instances, les formats des valeurs…. Les prévisions de ces modules de mise en correspondance sont combinées par un méta-module de mise en correspondance en employant la somme pondérée (*Définition 18*). Le résultat final des correspondances sera déduit à partir des valeurs de similarité agrégées en employant la technique d'optimisation de contraintes « relaxation labeling » (la technique permettant de résoudre le problème d'assignement des étiquettes aux nœuds d'un graphe en donnant un ensemble de contraintes). Un inconvénient de cette approche

est qu'elle se fonde principalement sur les instances des ontologies, qui ne sont pas toujours abondamment disponibles pour plusieurs ontologies. Un autre inconvénient est que l'ontologie est modélisée comme une taxonomie des concepts et que chaque concept a quelques attributs. Avec cette organisation, GLUE n'emploie pas des informations contenues dans la taxonomie (hiérarchie) des relations. GLUE fait également l'utilisation modeste des informations sur la taxonomie des concepts.

S-Match [Giunchiglia *et al*., 2004] [Giunchiglia *et al*., 2005] est un algorithme et un système pour chercher sémantiquement des correspondances, basé sur l'idée d'employer le moteur de la satisfiabilité propositionnelle (SAT) [Giunchiglia et Shvaiko, 2003] pour le problème de mise en correspondance des schémas. Il prend comme entrée deux graphes des concepts (schémas), et produit en sortie des rapports entre les concepts tels que l'équivalence, overlapping, différence (mismatch), plus général ou plus spécifique. L'idée principale de cette approche est d'utiliser la logique pour coder le concept d'un nœud dans le graphe et d'appliquer SAT pour trouver des rapports. Le *concept à un nœud*, qui est alors transformé en formule propositionnelle, est la conjonction de tous les *concepts des étiquettes* des nœuds sur le chemin de la racine du graphe jusqu'au nœud en question. Le concept d'étiquette d'un nœud est construit en deux étapes : (i) la normalisation de l'étiquette : telle que la tokenization, la lemmatisation, et (ii) l'extraction du sens de l'étiquette normalisée (des lemmes) à partir de WordNet [Miller, 1995]. Ensuite, les relations sémantiques (l'équivalence, plus général, plus spécifique) entre deux étiquettes de deux schémas sont (i) calculées grâce aux « matchers », les modules qui calculent la similarité entre deux étiquettes en employant des mesures de similarité de base (*1.2.1.2*) telles que la similarité des préfixes, des suffixes, la distance d'édition (*Définition 6*), la similarité de n-gram ; ou bien (ii) déduites en employant des matchers qui exploitent la sémantique dans WordNet, la similarité entre des hiérarchies, la similarité entre des commentaires [Giunchiglia et Yatskevich, 2004]. Ces relations sémantiques sont aussi encodées en logique. Enfin, le rapport entre deux concepts qui doit être prouvé est également converti en formule propositionnelle. Le moteur SAT calcule sur l'ensemble de formules propositionnelles pour vérifier si le rapport supposé est vrai ou faux. Cela permet donc de déduire des correspondances entre deux ontologies.

Le travail présenté dans [Ehrig et Sure, 2004] est un travail inspiré de COMA [Do et Rahm, 2002] dans le domaine de base de données. Ehrig et Sure proposent l'approche NOM pour mettre en correspondance des ontologies. La représentation interne des ontologies est en format `RDF(S)`. Cette approche se base sur un ensemble de règles. Les 17 règles listées sont catégorisées en différentes couches, dans lesquelles les règles exploitent des caractéristiques des entités à comparer. Ces couches correspondent aux couches du Web sémantique proposées par Tim Berners-Lee pour l'architecture du Web sémantique [Berners-Lee, 2003], telles que la couche des entités (instances), la couche des réseaux sémantiques, la couche des logiques de description… Les règles sont ensuite transformées en des formules de calcul de la similarité. Des mesures de base employées pour calculer la similarité sont l'égalité des chaînes de caractères, la similarité des chaînes (*Définition 6*) sur des noms de concept, de relation, d'instance ; la similarité des ensembles (*Définition 16*) pour des ensembles des entités formés des sous-concepts/relations, des super-concepts/ relations, des concepts/relations voisins, des instances, des instances de propriété… (*1.2.1.3.2*). Les valeurs de similarité calculées sont ensuite agrégées par des méthodes de combinaison telles que la somme pondérée (*Définition 18*), la fonction de Sigmoid (une variante de la somme pondérée), des

techniques d'apprentissage. Enfin, les meilleures valeurs agrégées de similarité sont choisies par une des méthodes « cut-off » (telles que la méthode du seuil, du delta ou du pourcentage [Do et Rahm, 2002]) pour déterminer des entités correspondantes.

Marc Ehrig et Steffen Staab présentent l'algorithme QOM (Quick Ontology Mapping) dans [Ehrig et Staab, 2004]. Cet algorithme est un échange entre l'efficacité (la qualité d'alignement) et l'efficacité (la vitesse de trouver des alignements), il s'agit d'une variante optimisée de NOM [Ehrig et Sure, 2004]. Il est employé pour mettre en correspondance des ontologies « light-weight » (plutôt des thesaurus) telles que la hiérarchie des sujets de l'ACM, la structure des répertoires dans des ordinateurs personnels, le WordNet, ou l'UMLS. Ce sont des taxonomies ayant un nombre énorme de concepts ($>10^4$ concepts). Ehrig et Staab ont montré que leur QOM met en correspondance ces ontologies dans un délai acceptable sans sacrifier beaucoup la qualité du résultat final. Comme NOM, QOM représente des ontologies en `RDF(S)` et utilise aussi des mesures de similarité telles que l'égalité des chaînes de caractères, la similarité des chaînes (*Définition 6*) sur des noms de concept, de relation, d'instance ; la similarité des ensembles (*Définition 16*)… Cependant, QOM a quelques modifications en comparaison avec NOM pour réduire la complexité de calcul, donc le temps d'exécution, telles que des stratégies de sélection des candidats à comparer (par hasard, des candidats ayant des étiquettes à proximité dans la liste triée…) ; des ensembles à comparer sont aussi limités (par exemple, QOM ne compare que deux ensembles de concepts parentaux *directs* de deux instances, au lieu de *tous* les concepts ancestraux comme dans NOM).

OLA [Euzenat et Valtchev, 2004] est un algorithme pour aligner des ontologies représentées en `OWL`. Il essaie de calculer la similarité de deux entités dans deux ontologies en se basant leurs caractéristiques (leurs types : classe, relation ou instance, leurs rapports avec d'autres entités : sous-classe, domaine, co-domaine…) et de combiner les valeurs de similarités calculées pour chaque paire d'entités de manière homogène. La combinaison est la somme pondérée des valeurs de similarité de chaque caractéristique. Les poids sont associés suivant le type d'entité à comparer et ses caractéristiques. Ils sont mis dans une matrice des poids et sont prédéfinis avant l'exécution de l'algorithme. Les mesures de similarité de base employées dans l'algorithme sont l'égalité des chaînes des caractères pour des URIs des entités, des mesures de similarité des suffixes ou des chaînes des caractères (*1.2.1.2.1*) pour des étiquettes des entités, la similarité (l'égalité) des types des données. Pour la similarité entre deux ensembles, le cas très souvent rencontré dans `OWL` (par exemple, en comparant deux entités, l'algorithme exploite la similarité de deux ensembles d'entités qui sont sous-entités des entités en question), il utilise la mesure de similarité basée sur des correspondances (*Définition 15*). À partir des valeurs de similarité calculées par des mesures de base, l'algorithme applique un calcul du point fixe, avec des itérations pour améliorer la similarité de deux entités. Quand il n'y a plus d'améliorations, des alignements entre deux ontologies sont générés.

Les travaux présentés dans cette section sont résumés dans le *Tableau 3*. Les relations entre les approches d'alignement des schémas et les approches d'alignement des ontologies avec les mesures de similarité utilisées (qui sont résumées dans la *Figure 5*) sont montrées dans la *Figure 6* et la *Figure 7*, respectivement. Nous voyons par exemple dans la *Figure 6*, Cupid emploie la technique du traitement automatique de la langue naturelle (normalisation du cas), la mesure basée sur des caractères (comparaison des chaînes de caractères), les synonymes des mots, la relation voisinage des entités… pour calculer la similarité de deux entités de deux schémas.

Approche	Entrée	Représentation interne	Mesures terminologiques	Mesures structurelles	Mesures extensionnelles	Mesures sémantiques	Mesures combinées	Observation*
MULTIKAT [Dieng et Hug, 1998a]	Ontologies personnelles	Graphe conceptuel	égalité des chaînes de caractères synonymes	Super-types Sous-types	-	-	Somme pondérée	
Anchor-PROMPT [Noy et Musen, 2001]	Ontologies	RDF(S), graphe RDF	égalité des chaînes de caractères pour les étiquettes des concepts et des relations	-	-	-	-	Comparaison des chemins (information structurelle) entre des paires d'ancres
Cupid [Madhavan *et al.,* 2001]	Schémas génériques	Graphes	Normalisation, catégorisation pour les noms d'élément de schéma Similarité des préfixes, des suffixes Thésaurus des synonymes, des hypernymes	Similarité des arbres, des feuilles	-	-	Somme pondérée	La valeur de similarité agrégée dépasse un seuil prédéfini
Similarity Flooding (SF) [Melnik *et al*., 2001] [Melnik *et al*., 2002]	Graphes génériques Schémas de SQL, XML	Graphes étiquetés orientés	Similarité des préfixes, des suffixes pour les noms des nœuds	-	-	-	-	Calcul du point-fixe : la similarité se répand sur le graphe (information structurelle)
GLUE [Doan *et al.,* 2002]	Ontologies + instances	Non précisée	Techniques terminologiques : tokenisation, stem égalité des chaînes de caractères pour des tokens	-	-	-	Méta-learner : somme pondérée	Apprentissage automatique (Bayes naïf) à partir des valeurs textuelles, des noms des instances
COMA [Do et Rahm, 2002]	Schémas : bases des données relationnelles, XML	Graphes acycliques orientés	Similarité des préfixes, des suffixe, de n-gram, des synonymes, des types de donnée, distance d'édition, soundex,	Similarité agrégée des enfants, des feuilles ou des chemins entre des éléments	-	-	La valeur maximale ou minimale, la somme pondérée ou la valeur moyenne	MaxN, MaxDelta, ou dépasse un seuil

S-Match [Giunchiglia *et al*., 2004] [Giunchiglia *et al*., 2005]	Schémas des bases des données, de XML, hiérarchies des concepts, ontologies	Graphes	Similarité des préfixes, des suffixe, de n-gram, la distance d'édition pour les étiquettes	Distance des hiérarchies	-	Encode le concept d'un nœud en logique propositionnelle. Formule un ensemble d'équations logiques. Vérifie par SAT	-	-
NOM [Ehrig et Sure, 2004]	Ontologies	RDF(S)	égalité des chaînes de caractères pour les URIs des concepts, des relations ou des instances similarité des chaînes pour des étiquettes des concepts, des relations ou des instances	Similarité des ensembles pour des super-concepts/relations, des sous-concepts/relations, des voisins, des concepts parentaux des instances	Similarité des ensembles pour des instances des concepts, des instances des relations	-	Somme pondérée Fonction de Sigmoid Technique d'apprentissage (trois niveaux) avec des réseaux neurologiques	-
QOM [Ehrig et Staab, 2004]	Ontologies « light-weight » : hiérarchie des sujets de l'ACM, structure des répertoire dans des ordinateurs personnels, le WordNet, l'UMLS	RDF(S)	Comme NOM	Comme NOM avec des modifications : limitations sur des voisines directes	Comme NOM	-	Comme NOM	-
OLA [Euzenat et Valtchev, 2004]	Ontologies en OWL	OL-graphes	égalité des chaînes de caractères pour les URIs des entités Similarité des chaînes des caractères pour des étiquettes (non précise) Similarité des types des données	Oui	-	-	Somme pondérée	Calcul du point-fixe (information structurelle)

* Technique principale utilisée pour déduire des correspondances

Tableau 3 Résumé des approches d'alignement de schéma et d'ontologie

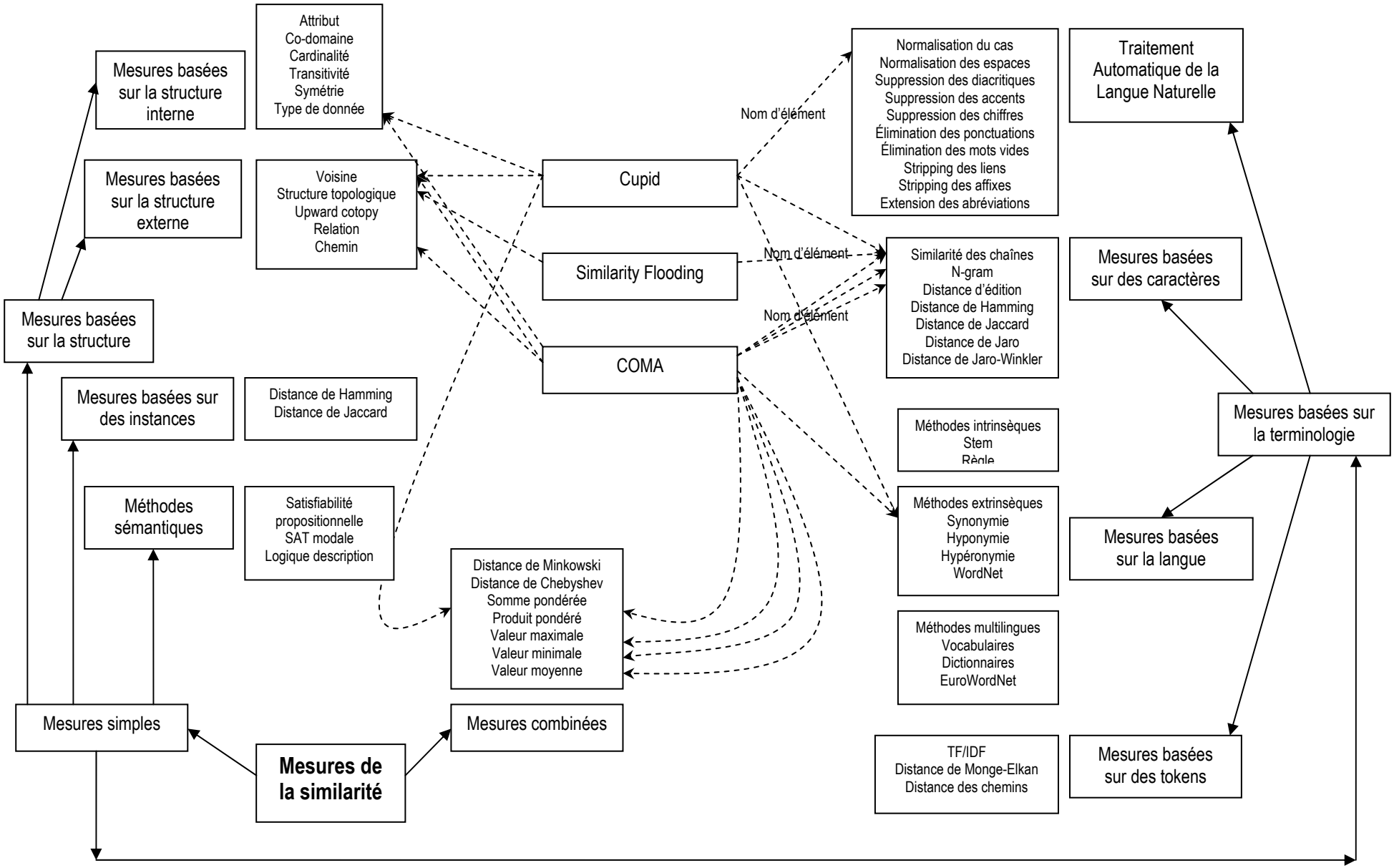


Figure 6 Des approches d'alignement des schémas et leurs relations avec des mesures de la similarité

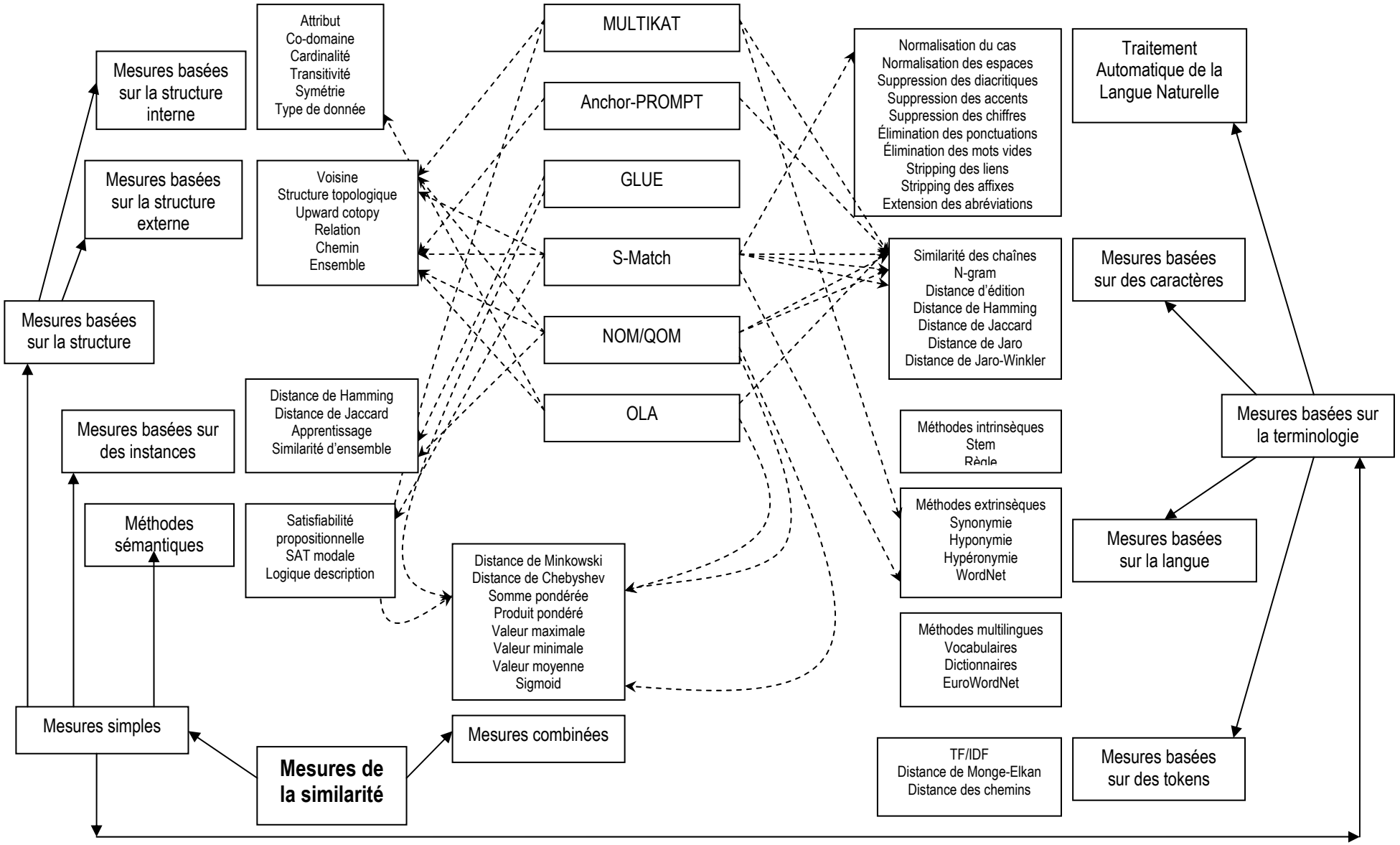


Figure 7 Des approches d'alignement des ontologies et leurs relations avec des mesures de la similarité

1.3 Point de vue

Un « *point de vue* », au sens propre, est un endroit depuis lequel nous observons ou regardons des choses : le paysage, des animaux, un arbre, une voiture… Il est clair que nous ne pouvons voir que la surface de l'objet qui s'oriente vers nous. Ainsi, nous ne pouvons examiner que les caractéristiques de cette surface de l'objet observé. Du sens propre au sens figuré, la notion de point de vue porte une image analogue : un point de vue est l'expression d'une opinion sur un sujet donné. Cette opinion peut être individuelle ou collective mais les caractéristiques du sujet ou de l'objet constatées par l'expression du point de vue sont toujours une partie de toutes les caractéristiques du sujet donné. Les caractéristiques relevées par un point de vue sont donc pertinentes et valides selon ce point de vue. Elles dépendent du lieu, de l'environnement, du climat… dans le cas du sens propre et de la personne, de son niveau de connaissances, de sa compétence, de son travail… dans le cas du sens figuré. Les expressions énoncées, donc les caractéristiques, selon différents points de vue sur un même sujet ou objet, peuvent être différentes voire contradictoires. Le point de vue d'une personne peut aussi changer lorsqu'elle change sa position, son environnement.

Le point de vue d'une personne sur un objet définit un contexte ou une situation, où la personne exprime ses expressions sur l'objet donné. Ces expressions sont valides et vraies selon le point de vue. Le point de vue a des propriétés principales suivantes : (1) la propriété de *filtrage* : un certain nombre des caractéristiques de l'objet sont considérées selon un point de vue ; (2) la propriété de l'*individu* : les énoncés d'une personne à propos des caractéristiques de l'objet sont considérées comme vraies pour cette personne-là, selon son point de vue ; et finalement, (3) la propriété de la *représentation* : cette propriété peut être considérée comme une combinaison de deux propriétés précédentes, elle est une représentation de l'objet dans le monde réel sous les yeux (la vue) d'une personne.

La représentation des connaissances dans un domaine consiste à modéliser et formaliser les connaissances existantes dans ce domaine. Si une ontologie utilisée dans la tâche de représentation des connaissances, elle doit être construite de manière la plus consensuelle et la plus générale possible. Les objets, les concepts dans l'ontologie doivent être bien sélectionnés et pertinents à représenter. Les termes représentant les objets, les concepts sont aussi bien choisis, ils devraient être clairs, non confus et leurs significations principales devraient correspondre aux objets, aux concepts à définir. De même, les caractéristiques de l'objet à décrire doivent aussi être soigneusement choisies parmi celles considérées comme les plus utiles et pertinentes dans le domaine. Tous ces choix et ces sélections sont effectués par des cogniticiens ou des ingénieurs d'ontologie et selon un *point de vue neutre* (pour ce domaine). Les points de vue des personnes qui participent à la représentation des connaissances, à la construction de l'ontologie doivent être rendus *neutres* ou *accordés* pour atteindre un certain niveau de consensus. Les points de vue des utilisateurs d'une ontologie sont supposés s'accorder avec le point de vue neutre selon lequel l'ontologie était construite.

Dans les sections suivantes, nous examinons quelques travaux dans le domaine de la représentation des connaissances qui prennent en compte la notion de point de vue.

1.3.1 Représentation des connaissances par objets

Plusieurs systèmes de représentation des connaissances par objets ont été construits en intégrant la notion de point de vue : KRL (Knowledge Representation Language) [Bobrow et Winograd, 1977] ; l'extension de KRL pour la linguistique OWL II [Martin, 1979] et pour le développement de programme PIE (Personal Information Environment) [Goldstein et Bobrow, 1980] ; l'amélioration de KRL, LOOPS [Stefik et Bobrow, 1985], en considérant des objets composites dont les composants sont les différents points de vue ; VIEWS [David, 1987], ROME [Carré, 1989], son extension pour le langage de Frames FROME [Dekker, 1994], SHOOD [Nguyen *et al*., 1992], TROPES [Marino, 1993].

Le modèle de la représentation des connaissances par objets repose principalement sur (1) la notion de *classe*, qui représente un ensemble d'objets dans le monde réel ayant des caractéristiques communes et les classes sont organisées en une hiérarchie avec des relations de subsomption entre elles; et (2) la notion d'*héritage* : les objets appartenant à une classe héritent de toutes les caractéristiques (attributs) de cette classe ; les sous-classes héritent de toutes les caractéristiques des classes ancêtres. La notion de point de vue est intégrée dans ce modèle sous forme de multi-héritage : un objet (une instance) est associé (hérité) à plusieurs classes selon différents points de vue (appelés perspectives en KRL).

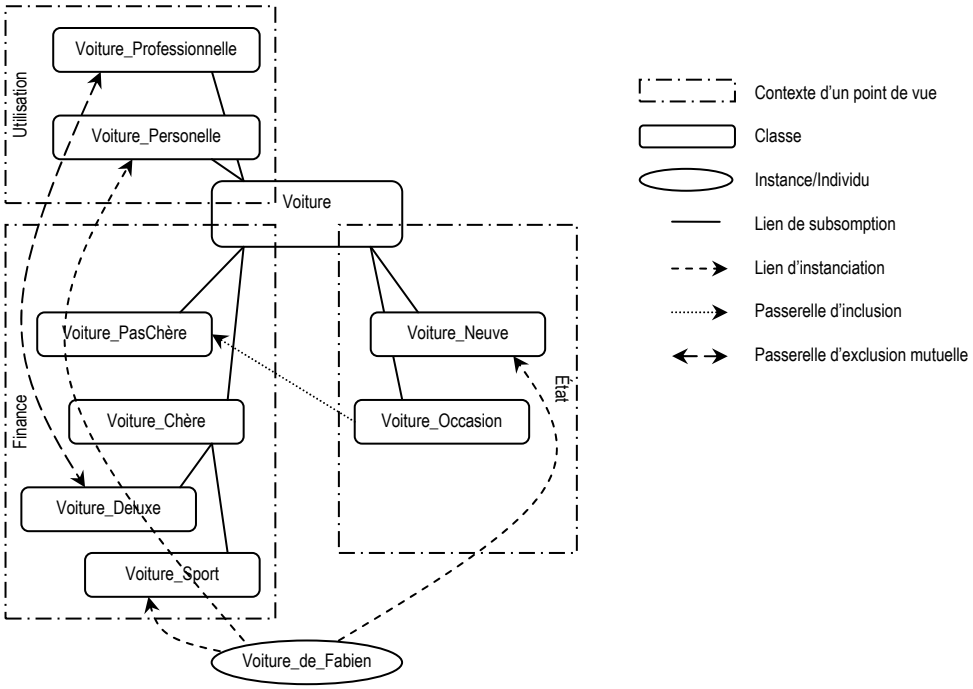


Figure 8 Exemple de modélisation avec TROPES

Parmi les systèmes de représentation des connaissances par objets (RCO) avec multi-points de vue citrons le système TROPES [Marino, 1993]. Dans son approche, à

part des éléments classiques de la RCO tels que classe, instance, attributs et facettes, TROPES emploie deux autres en plus : le concept et le point de vue. Chaque concept, qui correspond à un ensemble d'individus (objets dans le monde réel), est associé à un ou plusieurs points de vue. Les concepts sont disjoints, ils ne partagent pas les mêmes objets. La combinaison de concept et point de vue donne naissance à une hiérarchie de spécialisation de classes (sous forme d'un arbre), qui sont définies selon le point de vue et à propos du concept en question. Une classe ne peut donc spécialiser qu'une seule autre classe (pas de multi héritage). Un objet (instance) peut instancier plusieurs classes de différents points de vue (multi représentation). Entre des classes appartenant aux différents points de vue d'un même concept, il y a des liens, appelés *passerelles*. Cela permet d'exprimer des relations ensemblistes entre des ensembles d'instances possibles des classes de points de vue différents, et permet d'effectuer des raisonnements entre des points de vue tels que la vérification de la cohérence (une instance définie dans un point de vue doit satisfaire toutes les contraintes de ce point de vue, et aussi celles définies dans un autre point de vue si ces deux points de vue sont connectés par une passerelle). Un exemple de ce modèle est donné dans la *Figure 8*.

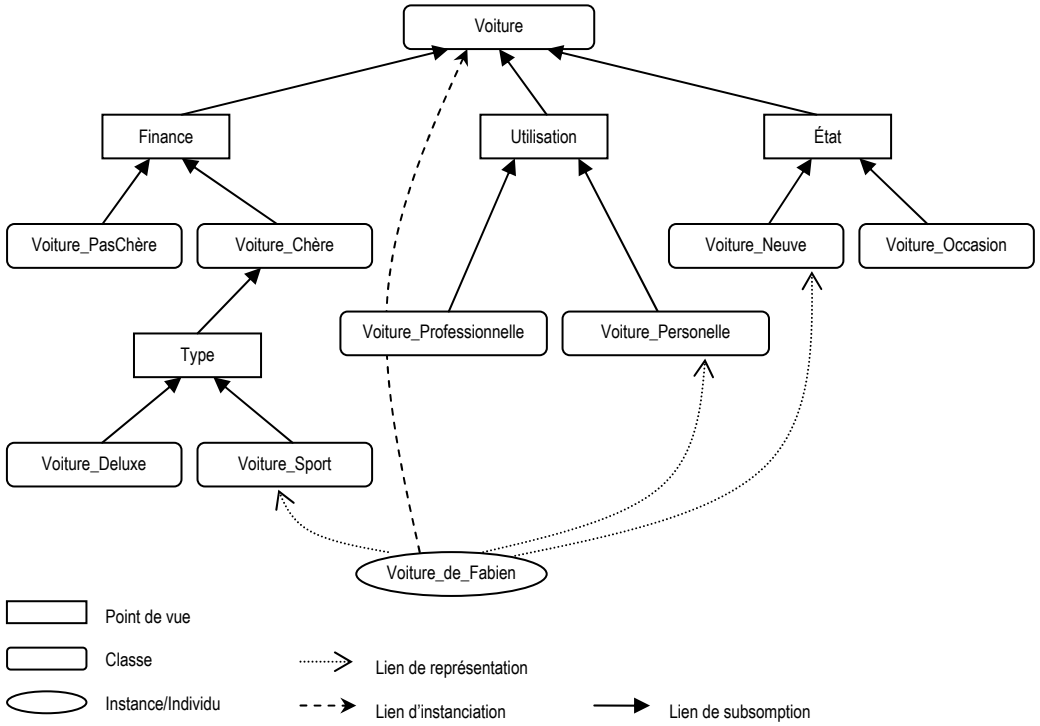


Figure 9 Exemple de modélisation avec ROME

Le système ROME [Carré, 1989] et son extension pour le langage de Frames FROME [Dekker, 1994] diffèrent de TROPES au niveau de la flexibilité dans le modèle de la représentation des objets. Tandis que dans TROPES, les points de vue sont prédéfinis et fixés pendant la construction du modèle, ROME permet d'ajouter de nouveaux points de vue et aussi de nouvelles représentations de l'objet pendant son utilisation via des *liens de représentation*. Cela permet d'avoir un système de représentation d'objets incomplets et évolutifs : les définitions des objets sont précisées,

modifiées ou changées lorsqu'ils évoluent dans le temps. Dans ce modèle, un objet est une instance d'une unique classe, dite *classe d'instanciation*, notée I-Classe et peut être relié via des liens de représentation à plusieurs autres classes, dites *classes de représentation*, notées R-Classe. La notion de lien de représentation permet de faire évoluer l'objet en ajoutant une nouvelle (resp. en supprimant une ancienne) représentation de l'objet sans changer la nature de l'objet qui est identifiée par son lien d'instanciation à la classe d'instanciation I-Classe. Un exemple de ce modèle est donné dans la *Figure 9*.

1.3.2 Représentation des connaissances par graphes

La notion de point de vue est intégrée dans le système VIEWS [Davis, 1987] sous la forme de *vue*. Le modèle proposé permet de représenter des objets structurés en décrivant plusieurs vues contenant des éléments communs. Une vue est définie par un réseau de parties, de relations et de contraintes. Cela permet de délimiter des relations qu'un objet peut posséder. Comme TROPES, ce système prédéfinit les vues (points de vue) via lesquelles un objet peut être examiné, les utilisateurs du système ne peuvent pas ajouter ou modifier ces vues.

Les travaux de Ribière [Ribière, 1999], [Ribière et Dieng, 1997, 2002] permettent d'utiliser le formalisme des graphes conceptuels [Sowa, 1984] pour représenter et gérer des points de vue explicitement dans une base des connaissances organisée selon de multiples points de vue. Le formalisme des graphes conceptuels proposé par Sowa se compose des éléments de base tels que des concepts, des relations, des types de concept, des types de relation. Les types de concepts ou de relations sont organisés dans un treillis (une hiérarchie) avec des liens sous-type entre eux. Ce type de lien est exploité et étendu dans ces travaux pour représenter et intégrer la notion de point de vue. Ainsi, nous pouvons décrire qu'un type de concept est sous-type d'un autre type de concept selon un point de vue, et dans ce cas-là, ce dernier est appelé le type de concept basique, le premier est appelé le type de concept v-orienté (orienté point de vue). Comme le modèle de ROME, ce modèle permet aussi d'exprimer des représentations multiples d'un concept : un concept instancie un type de concept basique et est relié à plusieurs concepts v-orientés via des liens de représentation. Comme le modèle de TROPES, ce modèle permet d'avoir des passerelles entre des concepts définis selon de différents points de vue.

1.3.3 Contexte et point de vue

La notion de *contexte* dans le domaine de la représentation des connaissances est très proche de la notion de point de vue. Le travail dans [Benerecetti *et al*., 2001] montre trois différentes formes du contexte dans la représentation des connaissances : (1) un contexte correspond à une partie, une vue *partielle* du domaine. La représentation des connaissances dans un contexte ne couvre donc qu'un sous-ensemble de connaissances dans le domaine ; (2) un contexte est une *approximation*. Les représentations des connaissances dans différents contextes ont différents niveaux d'approximation, différents niveaux de granularité ou d'abstraction ; (3) un contexte correspond à une

perspective. Les représentations des connaissances dans différents contextes, donc différentes perspectives, dépendent des éléments extérieurs tels que le lieu, le moment…

Dans notre vue sur la notion de point *de vue*, il semble que nous pouvons aussi exploiter les points de vue pour (1) « découper » des connaissances dans un domaine en plusieurs parties ; (2) séparer des connaissances de différents niveaux de granularité en différents points de vue ; ou bien (3) délimiter des connaissances selon leurs dépendances envers des éléments extérieurs par différents points de vue. Cela veut dire qu'il y a une correspondance forte entre ces deux notions : contexte et point de vue. Ainsi, les travaux concernant la construction du modèle de représentation multi-contextes peuvent aussi être exploités pour représenter de multi-points de vue.

Dans le cadre de la logique, John McCarthy présente dans [McCarthy, 1993] un modèle permettant la représentation et le raisonnement multi-contextes : toutes les assertions A de la logique sont associées à un contexte, sous forme c : A (où c correspond à un contexte), les propositions p sont aussi associées à un contexte, notées *ist*(c, p), indiquant que p est considérée comme vraie dans le contexte c, et les raisonnements sont réalisés dans ce contexte. Les combinaisons sont aussi possibles : si l'assertion *ist*(c, p) ci-dessus est valide dans le contexte c', nous écrivons c' : *ist*(c, p), et ainsi de suite.

Les travaux dans [Giunchiglia et Serafini, 1994], poursuivis ensuite dans [Ghidini et Giunchiglia, 2001] reposent sur la notion de contexte dans [Benerecetti *et al*., 2001] pour présenter les systèmes multi-contextes (MCS – Multi-Context Systems) et les sémantiques à modèles locaux (LMS – Local Models Semantics). Ces travaux introduisent des règles d'inférences pour faire des raisonnements au sein des contextes et aussi entre des contextes via des *passerelles* connectant les contextes locaux.

1.3.4 Ontologies multi-points de vue

[Falquet *et* Mottaz, 2001] et [Falquet *et* Mottaz, 2002] présentent une construction de l'ontologie multi-points de vue où les concepts sont associés à plusieurs définitions formelles correspondant aux différents points de vue sur les concepts en question. Les concepts sont organisés dans une hiérarchie et leurs places dépendent de leurs définitions. Le modèle proposé permet aussi d'avoir de multiples représentations des concepts et de multiples hiérarchies selon de différents points de vue.

Au lieu de représenter des concepts dans une ontologie selon plusieurs points de vue par un modèle de l'ontologie multi-points de vue, le formalisme `C-OWL` proposé dans [Bouquet *et al*., 2003] [Bouquet *et al*., 2004] essaie de *contextualiser* des ontologies en `OWL` pour intégrer la notion de point de vue dans les ontologies du Web sémantique. Ce formalisme est le résultat des travaux précédents de la représentation multi-contextes [Giunchiglia et Serafini, 1994] et [Ghidini et Giunchiglia, 2001]. Le langage `C-OWL` est une extension du langage `OWL` mais possède sa propre interprétation (inspirée de l'interprétation de `OWL`). Les ontologies en `OWL` sont mises dans des contextes, donc contextualisées, de `C-OWL`, et deviennent des *ontologies locales* en `C-OWL`. Les entités de ces ontologies locales (classes, relations, instances…) peuvent être reliées par des *passerelles*. Par exemple, un concept x de l'ontologie indexée (identifiée) par i est noté i :

x. Si ce concept x est plus spécifique que le concept y dans l'ontologie indexée par j, nous notons par une passerelle $i:x \xrightarrow{\subseteq} j:y$. Les passerelles avec leurs sémantiques bien prédéfinies en C-OWL permettent des raisonnements au niveau global, entre des ontologies locales (contextes locaux).

[d'Aquin, 2005] présente une application et des extensions des principes et des technologies de Web sémantique pour la construction d'un portail dédié à la gestion des connaissances en cancérologie. Ce travail emploie le langage C-OWL [Bouquet *et al*., 2003] [Bouquet *et al*., 2004] afin d'établir une représentation multi-points de vue des connaissances contenues dans les référentiels (sortes de protocoles de décision médicaux) du domaine de cancérologie, où il existe différents points de vue, différentes façons de considérer les connaissances du domaine et différentes façons de les utiliser. Les inférences fournies par C-OWL sont utilisées dans un mécanisme de raisonnement à partir de cas (RÀPC) décentralisé pour l'aide à la décision dans le cadre de l'application des référentiels.

1.4 Conclusion

Dans ce chapitre, nous avons présenté les connaissances de base concernant notre domaine de recherche. Il s'agit des principes et des technologies du Web sémantique : l'ontologie, l'architecture du Web sémantique structurée en couches, les langages du Web sémantique tels que XML, RDF(S), OWL. Nous avons montré que l'ontologie est un des composants les plus importants dans le Web sémantique.

Dans le but de résoudre les problèmes concernant l'hétérogénéité, et plus concretement l'hétérogénéité des ontologies, nous avons présenté des travaux dans la littérature qui abordent le problème d'alignement d'ontologies. Nous avons montré que les schémas ou les taxonomies peuvent être considérés comme des ontologies simples, et donc que les travaux dans le domaine d'alignement des schémas peuvent être adaptés pour mettre en correspondance les ontologies. Nous avons aussi présenté la notion de similarité entre deux entités dans le monde du Web sémantique utilisée dans notre contexte de recherche, ainsi que des méthodes de base permettant de calculer cette valeur de similarité. Pour une vue globale, nous avons fourni un tableau comparatif des approches d'alignement d'ontologies et des schémas, ainsi que deux figures montrant les relations entre les approches présentées et les mesures de similarité que ces approches emploient.

Toujours dans le but de permettre la construction et l'exploitation d'un Web sémantique dans une organisation hétérogène où l'on a besoin de résoudre les problèmes concernant l'hétérogénéité, nous avons présenté la notion de point de vue et quelques travaux dans le domaine de la représentation des connaissances qui prennent en compte cette notion. Nous avons montré que l'introduction de la notion de point de vue dans les systèmes de représentation des connaissances permet de cohabiter avec l'hétérogénéité. Enfin, nous avons présenté (1) deux approches permettant de construire des ontologies multi-points de vue et de contextualiser des ontologies ; (2) une approche employant la représentation des connaissances multi-points de vue dans une application réelle.

Chapitre 2

Algorithme d'alignement pour des ontologies en RDF(S)

Dans ce chapitre, nous présentons l'algorithme, nommé ASCO1, pour mettre en correspondance deux ontologies représentées en RDF(S). L'algorithme exploite des caractéristiques de langage RDF(S) pour déduire la similarité entre deux entités de deux ontologies. La valeur de similarité est calculée en deux parties : la partie linguistique qui se compose du nom, des étiquettes, des commentaires de l'entité, et la partie structurelle qui se base sur des rapports entre des entités et leurs places dans les taxonomies d'entité. Dépendant la valeur de similarité calculée, le rapport entre deux entités de deux ontologies en RDF(S) est décidé.

2.1 Alignement des Ontologies en RDF(S)

Comme nous l'avons expliqué dans la section *1.2.3*, la tâche d'alignement de deux ou plusieurs ontologies consiste à chercher des *rapports* entre des *entités* des ontologies en question, sans modifier ces ontologies. Les entités d'une ontologie peuvent être les classes, les relations, les fonctions, les instances, les axiomes, les règles… Les types de rapport possibles entre deux entités de deux ontologies peuvent être l'équivalence, la subsomption, l'exclusion ou l'incompatibilité. Le résultat de la tâche d'alignement est un ensemble de paires d'entités, qui sont des correspondances entre les ontologies. Ce résultat peut être employé pour effectuer plusieurs autres tâches émergeant dans le Web sémantique telles que la fusion des ontologies, la gestion des versions d'une ontologie, le partage de la connaissance, l'intégration sémantique …

Le modèle de méta données pour référer des ressources avec son schéma, `RDF(S)`, peut être employé pour représenter des ontologies simples. Le langage `RDF(S)` avec sa syntaxe en XML permet de décrire des entités des ontologies telles que des classes, des relations ou des instances. Dans notre contexte pour construire un algorithme d'alignement de deux ontologies en `RDF(S)`, nous considérons des ontologies simples avec les entités suivantes :

Classe (appelée aussi *concept*): Une classe est une représentation d'une collection ou d'un groupe d'objets ayant des caractéristiques similaires. Par exemple, une classe « `Voiture` » pourrait être employée pour représenter des voitures réelles dans le monde telles que la voiture Peugeot 205 de Khaled, la Twingo de Rose… Les classes dans une ontologie sont organisées dans une structure hiérarchique, avec des liens de subsomption ou de spécialisation entre elles. En `RDF(S)`, ce type de lien est représenté par la primitive `rdfs:subClassOf`. Une classe qui est sous-classe d'une autre, hérite toutes les caractéristiques de la classe mère. Par exemple, nous pouvons définir que la classe « `Véhicule` » est la classe mère de la classe « `Voiture` » car toutes les voitures sont des véhicules, elles héritent de toutes les caractéristiques d'un véhicule. De même, la classe « `Voiture` » peut avoir ses sous-classes telles que « `Voiture_Renault` », « `Break` », « `Berline` »…

Relation: Une relation est employée pour décrire un rapport entre deux classes. Ces deux classes sont indiquées par le *domaine* (domain en anglais) et le *co-domaine* (range en anglais) de la relation. Par exemple, la relation « `conduit` » pourrait être employée pour représenter le rapport entre la classe « `Personne` », qui est indiquée comme son domaine, et la classe « Véhicule », qui est son co-domaine. Comme les classes, les relations peuvent aussi être organisées dans une hiérarchie de relations, avec les liens de spécialisation représentés en `RDF(S)` par la primitive `rdfs:subPropertyOf`. Une relation qui est sous-relation d'une autre relation, hérite toutes les caractéristiques de la relation mère. Par exemple, la relation « `aime` » peut être définie comme sous-relation de la relation « `aAmi` », son co-domaine est la classe « `PetitAmi` » qui est sous-classe de la classe « `Personne` », le co-domaine de la relation mère « `aAmi` ».

Instance: Une instance d'une classe correspond à un objet réel qui est un membre de l'ensemble dénoté par cette classe. L'instance d'une classe possède toutes les caractéristiques définies pour la classe. Par exemple, « `Twingo_de_Rose` » peut être défini comme une instance de la classe « `Voiture` ».

Cependant, les ontologies représentables en `RDF(S)` ne sont que les ontologies « simples ». La sémantique exprimée par ces ontologies est limitée du faire de la puissance d'expression du langage `RDF(S)`. En `RDF(S)`, nous pouvons décrire et définir les concepts (les classes) en précisant leurs noms (en employant la primitive `rdf:id`), leurs étiquettes (`rdfs:label`), leurs descriptions ou commentaires (`rdfs:comment`), leurs classes parentales (`rdfs:subClassOf`) mais nous ne pouvons pas indiquer que deux classes quelconques sont équivalentes (toutes les instances d'une classe sont les instances d'autre et vice versa), ou que deux autres classes sont disjointes (aucune instance commune partagée). De même pour les relations de l'ontologie en `RDF(S)`, nous ne pouvons que décrire leurs noms, leurs étiquettes, leurs commentaires, leurs relations parentales, leurs domaines (`rdfs:domain`), leurs co-domaines (`rdfs:range`) mais il n'y a aucun moyen ou primitive permettant d'exprimer la cardinalité d'une relation, ou le fait qu'une relation est transitive, symétrique ou anti-symétrique, réflexive ou irréflexive, ou que deux relations sont équivalentes ou qu'une relation est l'inverse d'une autre… Pour les instances de l'ontologie, `RDF(S)` nous permet d'indiquer quelles classes elles instancient (en employant la primitive `rdf:type`) mais nous ne pouvons pas dire que deux instances sont les mêmes.

L'alignement de deux ontologies représentées en `RDF(S)` dépend donc des caractéristiques du langage `RDF(S)`. L'algorithme d'alignement exploite ces caractéristiques, la sémantique prédéfinie de `RDF(S)`, le contenu des ontologies pour déduire des correspondances entre les ontologies. Dans le contexte d'alignement de deux ontologies en `RDF(S)`, nous proposons un algorithme qui essaie d'exploiter le plus possible les caractéristiques des ontologies en `RDF(S)` telles que les noms d'entité, les étiquettes d'entité, les commentaires d'entité, les structures des ontologies, les rapports entre des entités dans une ontologie… pour mesurer la similarité (l'équivalence) entre deux entités de deux ontologies.

2.2 Fonctionnement de l'algorithme

L'algorithme que nous proposons et appelons ASCO1 pour faire aligner deux ontologies représentées en `RDF(S)`, prend comme entrée deux ontologies en `RDF(S)`, calcule automatiquement, sans interaction de l'utilisateur, des correspondances entre ces deux ontologies, et renvoie en sortie une liste des paires d'entités de deux ontologies indiquant la relation d'équivalence entre deux entités d'une paire, c'est-à-dire que, les deux entités d'une paire sont similaires (voir *Figure 10*).

Les ontologies d'entrée sont transformées en formalisme `RDF(S)` si elles sont dans un autre formalisme. Par exemple, les ontologies représentées en graphe conceptuel, en Topic Maps peuvent être transformées en `RDF(S)`. La transformation d'ontologies codées en graphe conceptuel vers `RDF(S)` et vice versa est réalisée dans [Corby *et al*., 2004]. Pour la transformation de Topic Maps vers `RDF(S)` (et vice versa), il existe

plusieurs travaux. [Garshol, 2003], [Garshol, 2003b] étudient le rapport entre les familles de standard Topic Maps et RDF. Ils comparent ces deux technologies et montrent comment convertir les informations de l'une à l'autre, comment convertir l'information de schéma, et comment effectuer des requêtes à travers ces deux représentations. [Pepper *et al.*, 2006] fait une synthèse des travaux concernant la transformation entre Topic Maps et RDF. Il s'agit des propositions de Moore, de Stanford, d'Ogievetsky, de Garshol, d'Unibo et d'autres. La transformation modifie le contenu de l'ontologie du fait des différences entre des formalismes. Cependant, le formalisme `RDF(S)` permet de représenter des ontologies simples ayant des éléments ontologiques de base tels que les classes, les relations, les instances ; de tels éléments sont partagées entre plusieurs formalismes. L'algorithme proposé ASCO1 se focalise sur exploitation de ces éléments de base dans son calcul de similarité, cela limite donc des influences des changements des connaissances ontologiques dans la transformation.

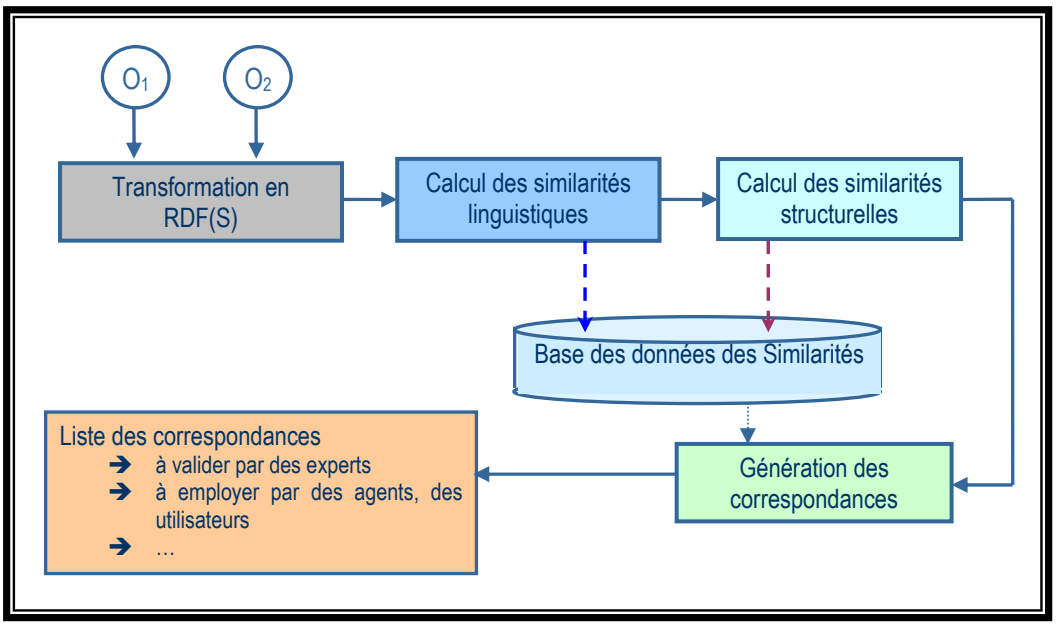


Figure 10 Le processus de l'algorithme d'alignement ASCO1

Les correspondances entre deux ontologies sont déduites à partir des similarités entre des entités de deux ontologies. L'algorithme effectue des calculs en se basant sur des caractéristiques communes de deux entités pour produire une valeur de similarité entre ces entités. Les calculs sont divisés en deux catégories de mesure selon la nature de la caractéristique exploitée de l'entité : (1) les calculs des similarités *linguistiques* (*2.3.1*) qui se basent principalement sur des aspects linguistiques de l'entité (*Figure 11*) tels que son nom, son étiquette, sa description… et (2) les calculs des similarités *structurelles* (*2.3.2*) qui exploitent les rapports d'une entité avec d'autres, sa position dans la structure hiérarchique des entités. Les valeurs de similarité calculées par des mesures linguistiques ou structurelles sont appelées les *valeurs de similarité partielles* entre deux entités, et elles sont stockées dans une *base des similarités*. Les valeurs de similarité partielles sont calculées par des mesures normalisées, elles sont donc comprises dans l'intervalle de 0 à 1. Ces valeurs de similarité partielles sont ensuite agrégées par une mesure de combinaison (la somme pondérée par des poids, *Définition 18*) pour atteindre une seule

valeur de similarité finale entre deux entités, qui est aussi comprise entre 0 et 1. En examinant cette valeur de similarité finale, deux entités sont considérées comme similaires (équivalentes) ou différentes.

2.3 L'algorithme en détail

En `RDF(S)`, les descriptions des classes et des relations dans l'ontologie ont beaucoup des points communs au niveau syntaxique. Elles sont décrites en utilisant des primitives de `RDF(S)` tels que `rdf:id`, `rdfs:label`, `rdfs:comment` pour leurs noms, leurs étiquettes, leurs commentaires respectivement. Les classes et les relations sont également organisées dans des hiérarchies (la hiérarchie de classe et la hiérarchie de relation) grâce aux deux primitives `rdfs:subClassOf` et `rdfs:subPropertyOf` respectivement. Quand à la différence entre une classe et une relation dans leurs descriptions en `RDF(S)` au niveau syntaxique, les deux primitives `rdfs:domain` et `rdfs:range` peuvent être utilisées pour spécifier le domaine et le co-domaine d'une relation. L'algorithme d'alignement ASCO1 exploite les éléments communs à partir des descriptions des classes et des relations pour mesurer la similarité entre deux classes et deux relations, respectivement. Donc, dans ce chapitre, nous présentons des mesures de similarité pour les classes, la similarité entre les relations est calculée par des mesures construites de la même manière.

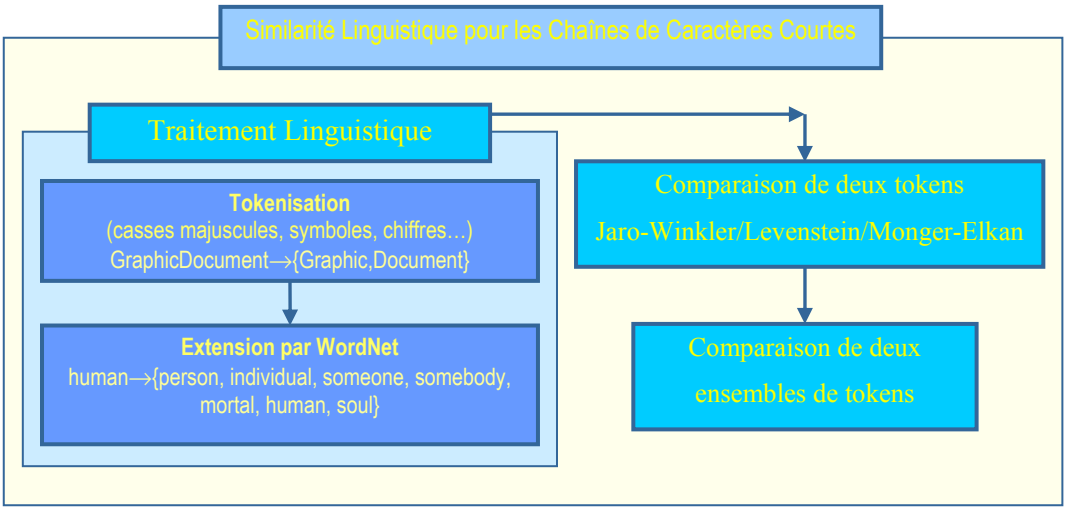


Figure 11 Les calculs des similarités linguistiques pour des chaînes de caractères courtes

2.3.1 La similarité au niveau de linguistique

La similarité linguistique de deux classes est calculée à partir des composantes linguistiques de la classe. En `RDF(S)`, les composantes linguistiques dans une description d'une classe sont le nom de classe (spécifié par la primitive `rdf:id`), les étiquettes de la classe (spécifiées par une ou plusieurs primitives `rdfs:label`) et les commentaires de la classe (spécifiés par une ou plusieurs primitives `rdfs:comment`). Suivant la nature de

chaque composante linguistique, une mesure de similarité est construite pour pouvoir exploiter au maximum des informations contenues dans la composante.

Nous distinguons deux catégories des composantes linguistiques : les composantes linguistiques qui sont les *chaînes de caractères courtes* et celles qui sont les *chaînes de caractères longues*. La première catégorie se compose des noms de classe et des étiquettes qui existent normalement sous forme d'un mot, un terme, ou au maximum une expression de quelques mots. La deuxième catégorie contient des commentaires de la classe, qui sont souvent une expression, une phrase, ou voire un paragraphe. Les mesures de similarité proposées sont différemment conçues pour chaque catégorie.

2.3.1.1 La similarité des noms

En `RDF(S)`, des ressources, donc les classes ou les relations, sont identifiées par des URIs. Par exemple, la classe « `Article` » est référencée en `RDF(S)` par l'URI « http://www.inria.fr/acacia/exemple#Article »

```
<rdfs:Class rdf:about="http://www.inria.fr/acacia/exemple#Article" />
```

L'URI se compose de deux parties : l'espace de noms (namespace) et le nom local. Dans l'exemple ci-dessus, l'espace de noms de l'URI est « http://www.inria.fr/acacia/exemple », et le nom local est « `Article` ». Si deux ressources ont une même URI, elles sont exactement la même ressource. Alors, si deux classes de deux ontologies sont référencées par une même URI, elles sont parfaitement la même classe et donc elles sont similaires (la valeur de similarité entre elles est de 1). Cependant, l'alignement d'ontologies traite normalement différentes ontologies ayant différents espaces des noms, il est intéressant donc de ne comparer que les noms locaux des classes, appelé désormais pour simplifier les noms de classe.

Normalement, le nom d'une classe est une chaîne des caractères, sans espaces. Un nom de classe peut être un mot, un terme, ou une expression (une combinaison des mots). Ce nom est unique dans une ontologie pour identifier la classe. Le calcul de la valeur de similarité de deux noms est effectué dans deux étapes: *la normalisation* et *la comparaison*.

Algorithme 1 Normalisation(nom)

```
résultat <- créer un ensemble vide
tokens <- Tokenisation(nom)
n <- nombre de tokens dans tokens
Pour i de 1 à n
    ex <- Expansion(tokens[i])
    ex <- Minusculisation(ex)
    result <- Ajouter(ex, result)
Fin Pour
Retourner résultat
```

L'étape de normalisation (*Algorithme 1*) convertit un nom de classe en un ensemble d'unités lexicales des *tokens*. Un nom est découpé en plusieurs tokens grâce à la ponctuation, à la casse (majuscule), aux symboles spéciaux, aux chiffres. Par exemple, le

nom de classe « SpatialEntity » est découpé en deux tokens « Spatial » et « Entity » ; le nom « Voiture_de_Rose » est convertit à l'ensemble {« Voiture », « de », « Rose »}. La normalisation du nom inclut une expansion de token : les abréviations, les acronymes sont élargis, par exemple le token « WS » est élargi à { « Web », « Sémantique » }. Cette expansion est effectuée grâce à un dictionnaire externe, dont chaque entrée est une paire composée d'un token (abréviation ou acronyme) et d'un ensemble de mots qui correspondent au token. Le dictionnaire est soit construit spécialement pour le domaine où les ontologies à aligner se trouvent soit il s'agit d'un dictionnaire général contenant des termes communs. Les tokens dans l'ensemble de tokens sont enfin rendus minuscules pour être comparés après.

Algorithme 2 Sim_Max_Dans_Ensemble(token, tokens)

```
valeur_max <- 0
n <- nombre de tokens dans tokens
Pour i de 1 à n
    valeur_sim <- Mesure_Similarite(token, tokens[i])
    Si valeur_sim > valeur_max
        valeur_max <- valeur_sim
    Fin Si
Fin Pour
Retourner valeur_max
```

La comparaison de la similarité de deux noms de deux classes (*Algorithme 3*) est la comparaison entre deux ensembles de tokens correspondant à ces noms. La similarité de deux tokens, qui sont actuellement des chaînes de caractères courtes, est calculée en employant la métrique Jaro-Winkler (*Définition 8*), qui est basée sur le nombre et l'ordre de caractères communs entre deux chaînes des caractères [Winkler, 1999]. La *Définition 8* est la version de la mesure de distance de la métrique Jaro-Winkler, $DS_{Jaro\text{-}Winkler}$. Puisque la métrique Jaro-Winkler est une métrique normalisée, donc la mesure de similarité Jaro-Winkler $S_{Jaro\text{-}Winkler}$ est obtenue par : $S_{Jaro\text{-}Winkler} = 1 - DS_{Jaro\text{-}Winkler}$. La valeur de similarité de nom (S_{Nom}) entre deux noms est alors la moyenne des valeurs de similarité entre chaque token d'un ensemble et le token le plus similaire dans l'autre ensemble (*Algorithme 2*).

Définition 20 *(Similarité des noms). Soit n_1 et n_2 deux noms de deux classes. Soit N_1 et N_2 ensembles des tokens obtenus après la normalisation de n_1 et de n_2, respectivement. La similarité des noms est une fonction de la similarité S_{Nom} : $\mathcal{S} \times \mathcal{S} \rightarrow [0, 1]$ telle que :*

$$S_{Nom}(n_1, n_2) = \frac{\sum_{m_1 \in N_1} MJW(m_1, N_2) + \sum_{m_2 \in N_2} MJW(m_2, N_1)}{|N_1| + |N_2|}$$

$$\text{où } MJW(m_i, N) = \max_{m_j \in N} S_{Jaro-Winkler}(m_i, m_j)$$

$|N_i|$, i = 1, 2, est la cardinalité de l'ensemble N_i.

Algorithme 3 Similarité_des_Noms(nom1, nom2)

```
tokens1 <- Normalisation(nom1)
tokens2 <- Normalisation(nom2)
n1 <- nombre de tokens dans tokens1
n2 <- nombre de tokens dans tokens2
somme1 <- 0
Pour i de 1 à n1
    sim <- Sim_Max_Dans_Ensemble(tokens1[i], tokens2)
    somme1 <- somme1 + sim
Fin Pour
somme2 <- 0
Pour j de 1 à n2
    sim <- Sim_Max_Dans_Ensemble(tokens2[j], tokens1)
    somme2 <- somme2 + sim
Fin Pour
Si n1 = n2 = 0
    résultat <- 1
Sinon
    résultat <- (somme1 + somme2) / (n1 + n2)
Fin Si
Retourner résultat
```

Dans la *Définition 20*, au lieu d'employer la métrique Jaro-Winkler pour calculer la similarité de deux tokens, l'algorithme permet de choisir d'autres métriques bien connues de comparaison des chaînes de caractères courtes telle que Levenstein, Monge-Elkan [Cohen *et al*., 2003]. Le *Tableau 1*, page 24, liste des domaines d'application pour quelques mesures de la similarité. Nous trouvons que la mesure Monge-Elkan donne un résultat le meilleur en général, dans la plupart des cas, même si le domaine d'application peut être non déterminé. Par contre, la mesure Jaro-Winkler a une performance presque celle de Monge-Elkan mais la vitesse du calcul est plus rapide. Dans le cas où on compare des entités numériques ayant des tailles fixes, la mesure Hamming sera le choix meilleur.

Par exemple, le nom de la classe « `primitive-type` » dans l'ontologie `java_prog101.rdfs` est normalisé à l'ensemble {« `primitive` », « `type` »}, le nom de la classe « `Primitive_Types` » dans l'ontologie `java_ontology.rdfs` est normalisé à l'ensemble {« `primitive` », « `types` »}. La valeur de la mesure de Jaro-Winkler pour paire (« `primitive` », « `primitive` ») est de 1,0 ; pour paire (« `type` », « `types` ») est de 0,96 ; pour paires (« `primitive` », « `type` ») et (« `primitive` », « `types` ») sont de 0,0. Donc, la valeur de similarité de ces deux classes est déduite de la comparaison de deux ensembles de tokens correspondants, calculée par la mesure de similarité des noms (*Définition 20*) : $S_{Nom} = ((1 + 0,96) + (1 + 0,96)) / (2 + 2) = 0,98$.

2.3.1.2 La similarité des étiquettes

Les étiquettes de classe sont également des chaînes des caractères. Tandis que le nom de classe sert à identifier de manière unique la classe dans l'ontologie et à être utilisé par les programmes informatiques, l'étiquette de classe est employée pour fournir une version lisible du nom de classe pour les gens. Ainsi, le calcul de la similarité entre des

étiquettes est semblable au calcul de la similarité des noms. Cependant, en RDF(S), une classe peut être associée à zéro, une ou plusieurs étiquettes. Cela permet, par exemple, d'avoir plusieurs versions lisibles dans plusieurs langues naturelles pour une classe.

```
<rdfs:Class rdf:about="http://www.inria.fr/acacia/exemple#Something">
    <rdfs:label xml:lang="en">thing</rdfs:label>
    <rdfs:label xml:lang="en">something</rdfs:label>
    <rdfs:label xml:lang="fr">chose</rdfs:label>
</rdfs:Class>
```

Algorithme 4 Sim_Max_Dans_Ensemble_Etiq(etiq, etiqs)

```
valeur_max <- 0
n <- nombre d'étiquettes dans etiqs
Pour i de 1 à n
    valeur_sim <- Similarité_des_Noms(etiq, etiqs[i])
    Si valeur_sim > valeur_max
        valeur_max <- valeur_sim
    Fin Si
Fin Pour
Retourner valeur_max
```

Le calcul de la similarité des étiquettes (*Algorithme 5*) entre deux ensembles d'étiquettes de deux classes est alors obtenu par extension du calcul de la similarité des noms (*Définition 20*). Pour chaque étiquette d'une classe, l'étiquette de l'autre classe la plus similaire (ayant la valeur de similarité calculée par S_{Nom} le plus élevée) est cherchée (*Algorithme 4*). La valeur de la similarité des étiquettes est la valeur moyenne de toutes les valeurs de similarité des noms des paires d'étiquettes trouvées.

Définition 21 *(Similarité des étiquettes). Soit $\mathcal{L}_1$ et $\mathcal{L}_2$ ensembles des étiquettes de deux classes de deux ontologies. La similarité des étiquettes est une fonction de la similarité $S_{Étiquette} : 2^E \times 2^E \rightarrow [0, 1]$ telle que :*

$$S_{Etiquette}(\mathcal{L}_1, \mathcal{L}_2) = \frac{\sum_{L_1 \in \mathcal{L}_1} MNS(L_1, \mathcal{L}_2) + \sum_{L_2 \in \mathcal{L}_2} MNS(L_2, \mathcal{L}_1)}{|\mathcal{L}_1| + |\mathcal{L}_2|}$$

$$\text{où } MNS(L_i, \mathcal{L}) = \max_{L_j \in \mathcal{L}} S_{Nom}(L_i, L_j)$$

$|\mathcal{L}_i|$, i = 1, 2, est la cardinalité de l'ensemble $\mathcal{L}_i$.

Dans le cas où toutes les deux classes n'ont pas d'étiquettes, $\mathcal{L}_1 = \mathcal{L}_2 = \varnothing$, $|\mathcal{L}_1| = |\mathcal{L}_2| = 0$, la similarité des étiquettes $S_{Etiquette}$ est considérée égale à 1.

Algorithme 5 Similarité_des_Etiquettes(etiqs1, etiqs2)

```
n1 <- nombre d'étiquettes dans etiqs1
n2 <- nombre d'étiquettes dans etiqs2
somme1 <- 0
Pour i de 1 à n1
    sim <- Sim_Max_Dans_Ensemble_Etiq(etiqs1[i], etiqs2)
    somme1 <- somme1 + sim
Fin Pour
```

```
somme2 <- 0
Pour j de 1 à n2
    sim <- Sim_Max_Dans_Ensemble_Etiq(etiqs2[j], etiqs1)
    somme2 <- somme2 + sim
Fin Pour
Si n1 = n2 = 0
    résultat <- 1
Sinon
    résultat <- (somme1 + somme2) / (n1 + n2)
Fin Si
Retourner résultat
```

2.3.1.3 La similarité des commentaires

En RDF(S), on peut associer à une classe (ou relation) de l'ontologie un ou plusieurs commentaires (descriptions). Ce sont des descriptions servant à fournir des informations descriptives de la classe aux personnes. En lisant les commentaires de la classe, les utilisateurs comprennent la notion et la signification du concept défini par la classe. Ainsi, ces informations peuvent être exploitées, par la comparaison des commentaires, pour déduire la similarité entre deux classes.

Un exemple de la classe « Something » avec ses commentaires en RDF(S) :

```
<rdfs:Class rdf:about="http://www.inria.fr/acacia/exemple#Something">
    <rdfs:comment xml:lang="en">Whatever exists animate, inanimate or
     abstraction.</rdfs:comment>
    <rdfs:comment xml:lang="fr">Tout ce qui existe anime, inanime ou
    abstraction.</rdfs:comment>
</rdfs:Class>
```

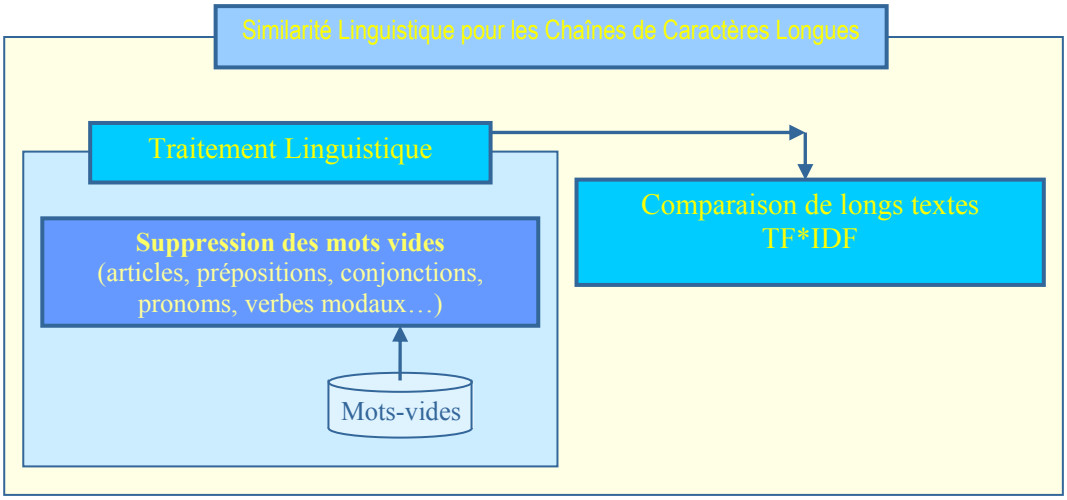


Figure 12 Les calculs des similarités linguistiques pour des chaînes de caractères longues

Les commentaires d'une classe, représentés en RDF(S) par la primitive `rdfs:comment`, sont décrits souvent sous forme d'un long texte comprenant des expressions, des phrases ou voire des paragraphes. Pour cette raison, la comparaison des commentaires en employant des mesures de similarité conçues pour s'occuper des chaînes de caractères courtes (des mots, des termes, des tokens), telles que Jaro, Smith-Waterman n'est plus appropriée. Au lieu de cela, nous proposons une mesure, qui se base sur la technique TF/IDF, une méthode bien connue et employée dans le domaine de la recherche d'information et de la classification, pour calculer la similarité entre les commentaires.

Dans son utilisation originale dans le domaine de la recherche d'information, la technique TF/IDF est employée pour effectuer des recherches. TF/IDF est une manière d'évaluer la pertinence d'un terme en rapport à un document (*Définition 9*). Pour une recherche dans un ensemble de documents, par exemple, chaque document est associé à un vecteur, dont les dimensions correspondent aux termes dans tous les documents. Les valeurs des dimensions de ce vecteur sont calculées par TF/IDF. Une requête est aussi représentée par un vecteur. La recherche est effectuée par les corrélations entre les vecteurs représentant les documents et le vecteur de requête.

De l'analogie entre la recherche d'un document dans un ensemble de documents et l'alignement d'ontologies qui est en fait la recherche de l'entité dans une ontologie la plus similaire à une autre entité d'une autre ontologie, nous proposons une mesure utilisant la technique TF/IDF pour calculer la similarité entre deux classes (ou deux relations, ou deux instances) de deux ontologies. Nous transformons et exprimons ce problème comme suit : les classes de l'ontologie sont considérées comme des documents dans le domaine de la recherche d'information. Les mots dans les commentaires d'une classe sont des mots du document. L'univers des documents est l'univers de classes qui est construit avec toutes les classes dans les deux ontologies. Chaque classe est associée à un vecteur. Les dimensions de ces vecteurs sont calculées par la technique TF/IDF avec l'univers de classes et les mots des commentaires. Ainsi, la similarité de deux classes est déduite de la similarité entre leurs commentaires qui s'obtient à partir de la distance entre deux vecteurs représentant deux classes en question.

Rappelons que la notion de « *similarité* » et celle de « *distance* » sont des notions inverses. Si deux entités sont similaires (la valeur de similarité entre elles est élevée), la distance entre elles est petite, et réciproquement, plus cette distance est lointaine, plus elles sont différentes, plus la valeur de dis-similarité est élevée, plus la valeur de similarité est petite. La notion de distance est donc appelée la dis-similarité ou la différence. Si la valeur de similarité S et celle de distance DS sont normalisées, nous avons $S + DS = 1.0$ (voir *1.2.1.1*).

Nous illustrons comment nous calculons ces vecteurs et la valeur de similarité de deux commentaires.

Algorithme 6 Construire_Univers(onto, U)

```
n1 <- nombre de classes dans l'ontologie onto
Pour i de 1 à n1
    une_classe <- prendre i-ème classe de l'ontologie onto
    cmt <- extraire le commentaire de la classe une_classe
    Supprimer_Mots_Vides(cmt)
    tokens <- Tokenisation(cmt)
    n_tokens <- nombre de tokens dans tokens
    Pour j de 1 à n_tokens
        Si l'univers U ne contient pas le token tokens[j]
            U <- Ajouter(tokens[j], U)
        Fin Si
    Fin Pour
Fin Pour
Retourner U
```

Soit O_1, O_2 deux ontologies à aligner. Soit U l'univers des mots, construit des mots contenus dans les commentaires de toutes les classes de O_1 et de O_2 (*Algorithme 6*). Soit S = |U| est le nombre des mots distincts dans l'univers U. Soit v = (w_1, w_2, … w_S) un vecteur représentant une certaine classe c (*Algorithme 7*). La valeur de la dimension i-ème w_i du vecteur est calculée par la technique TF/IDF comme suivante :

$$w_i = tf_i * idf_i$$

$$idf_i = \log_2 \frac{N}{n_i}$$

où tf_i (fréquence de terme) est le nombre de fois où le i-ème mot dans l'univers U apparaît dans le commentaire de la classe c, idf_i (fréquence inverse de document) est l'inverse du pourcentage des classes qui contiennent le mot w_i, N est le nombre de classes dans les deux ontologies O_1 et O_2, n_i est le nombre de classes qui contiennent le mot w_i au moins une fois.

Algorithme 7 Construire_Vecteur(entité, U)

```
S <- nombre de mots distincts dans l'univers U
vecteur <- créer une matrice de S éléments
N <- nombre de classes dans toutes les deux ontologies
Pour i de 1 à S
    n <- nombre de classes qui contiennent le mot U[i] au
moins une fois
    idf <- log2(N/n)
    tf <- nombre de fois où le i-ème mot dans l'univers U apparaît
dans le commentaire de l'entité entité
    vecteur[i] <- tf * idf
Fin Pour
Retourner vecteur
```

Notons qu'en RDF(S), une classe peut être décrite en employant plusieurs fois la primitive `rdfs:comment`. Elle a donc plusieurs commentaires. Cela permet de donner

des commentaires de la classe en plusieurs différentes langues. La langue est spécifiée par la primitive `xml:lang`. La comparaison de deux commentaires est effectuée sur deux commentaires en même langue. Dans le cas où la classe est définie avec plusieurs commentaires dans une même langue, ces commentaires sont alors agrégées (concaténer des textes) en un seul commentaire, qui est ensuite représenté par un vecteur pour le calcul de similarité de commentaire avec le commentaire d'une autre classe.

La similarité entre deux commentaires de deux classes se calcule donc à partir de « *distance* » entre deux vecteurs qui les représentent (*Algorithme 8*). Plus deux vecteurs sont proches (la distance petite), plus les deux commentaires correspondants sont similaires (la valeur de similarité élevé).

> ***Définition 22*** *(Similarité des commentaires). Soit* $v_i = (w_{i1}, w_{i2}, \ldots w_{iS})$ *et* $v_j = (w_{j1}, w_{j2}, \ldots w_{jS})$ *deux vecteurs représentant deux commentaires de deux classes de deux ontologies. La similarité des commentaires est une fonction de la similarité (coefficient de cosinus)* $S_{Commentaire}$ *:* $\mathcal{V}x\,\mathcal{V} \rightarrow [0, 1]$ *telle que :*

$$S_{Commentaire}(v_i, v_j) = \frac{\sum_{k=1}^{S} w_{ik} w_{jk}}{\sqrt{\sum_{k=1}^{S} (w_{ik})^2 * \sum_{k=1}^{S} (w_{jk})^2}}$$

Algorithme 8 Similarité_des_Commentaires(entité1, entité2)

```
U <- créer un ensemble vide
U <- Construire_Univers(ontologie1, U)
U <- Construire_Univers(ontologie2, U)
vecteur1 <- Construire_Vecteur(entité1, U)
vecteur2 <- Construire_Vecteur(entité2, U)

S <- nombre de mots distincts dans l'univers U
somme1 <- 0
somme2 <- 0
somme3 <- 0
Pour i de 1 à S
    somme1 <- somme1 + vecteur1[i] * vecteur2[i]
    somme2 <- somme2 + vecteur1[i] * vecteur1[i]
    somme3 <- somme3 + vecteur2[i] * vecteur2[i]
Fin Pour

résultat <- somme1 / racine(somme2 + somme3)
Retourner résultat
```

Comme nous en avons discuté, les commentaires des classes se composent habituellement d'un texte descriptif. Ainsi, une étape de prétraitement linguistique est nécessaire pour atteindre un meilleur résultat. Cette étape élimine tous les *mots vides* contenus dans les commentaires. Les mots vides sont des mots qui contiennent peu d'information utile, tels que les articles (the, a, an,…), les prépositions (to, of, in,…), les conjonctions (and, or,…), les pronoms (you, I,…), les verbes modaux (are, is, was,…). Nous employons une bibliothèque externe des mots vides pour cette étape. Certaines

bibliothèques de mots vides en différentes langues peuvent être trouvées sur l'Internet[1] : pour l'anglais[2], pour le français[3, 4], pour l'espagnol, le portugais, l'italien, le russe, le danois, le norvégien, le suédois, le hollandais, l'allemand. Nous pouvons aussi trouver une liste multilingue[5] des mots vides en l'anglais, le français, l'italien, l'espagnol et le portugais.

Dans le cas où toutes les deux classes n'ont pas de commentaires, la similarité des commentaires $S_{Commentaire}$ est considérée égale à 1.

2.3.1.4 Combinaison des similarités linguistiques

Trois valeurs de similarité, S_{Nom}, $S_{Etiquette}$, $S_{Commentaire}$, calculées à partir de trois composantes linguistiques de classe (le nom, les étiquettes, les commentaires) dans les sections ci-dessus sont agrégées pour produire une *valeur de similarité linguistique* de deux classes, nommé $S_{Linguistique}$ (*Algorithme 14*). L'agrégation est la somme pondérée (*Définition 18*) avec les poids w_{Nom}, $w_{Etiquette}$, $w_{Commentaire}$ choisis au début de l'algorithme, suivant la nature des ontologies à aligner : si ces ontologies partagent plusieurs classes ayant les mêmes noms, w_{Nom} est mis à une valeur plus élevée, par contre, si les classes sont bien décrites avec des commentaires en détail, le poids pour les commentaires $w_{Commentaire}$ a une valeur plus élevée.

> ***Définition 23*** *(Similarité linguistique de deux classes). Soit A et B deux classes de deux ontologies. Soit Nom(X), Etiquette(X) et Commentaire(X) les fonctions qui retournent le nom, les étiquettes, les commentaires de la classe X. La similarité linguistique de deux classes A et B est une fonction de la similarité* $S_{Linguistique}$ *:* $O \times O \rightarrow [0, 1]$ *telle que :*

$$S_{Linguistique}(A,B) = S_{Nom}(Nom(A),Nom(B)) * w_{Nom} + S_{Etiquette}(Etiquette(A),Etiquette(B)) * w_{Etiquette} + S_{Commentaire}(Comment(A),Comment(B)) * w_{Commentaire}$$

$$w_{Nom} + w_{Etiquette} + w_{Commentaire} = 1$$

Si cette valeur de similarité linguistique $S_{Linguistique}(A, B)$ excède un seuil prédéfini $T_{Linguistique} > 0$, nous considérons que la classe A de l'ontologie O_1 est linguistiquement similaire avec la classe B de l'ontologie O_2, et noté l-similar(A, B).

---

[1] http://snowball.tartarus.org/

[2] http://snowball.tartarus.org/algorithms/english/stop.txt

[3] http://snowball.tartarus.org/algorithms/french/stop.txt

[4] http://www.up.univ-mrs.fr/veronis/data/antidico.txt

[5] http://library.wur.nl/isis/docum.html

Définition 24 *(l-similar). Soit A et B deux classes de deux ontologies. Soit* $T_{Linguistique}$ *un nombre réel non négatif. Deux classes A et B sont linguistiquement similaires et noté l-similar(A, B) si et seulement si* $S_{Linguistique}(A, B) \geq T_{Linguistique}$.

2.3.1.5 L'intégration avec WordNet

WordNet [Miller, 1995] est un système de référence lexicologique (voir en plus *1.2.1.2.3*). Les noms, les verbes, les adjectifs et les adverbes en anglais sont organisés en des ensembles des synonymes. Ces ensembles des synonymes, appelés *synsets*, sont reliés par différentes relations sémantiques. Un système similaire, inspiré de la structure de WordNet, nommé EuroWordNet [EuroWordNet, 1999], est construit pour des langues européennes telles que le hollandais (44015 synsets), l'italien (40428 synsets), l'espagnol (23370 synsets), l'allemand (15132 synsets), le français (22745 synsets), le tchèque (12824 synsets) et l'estonien (7678 synsets) [1].

En utilisant WordNet, nous pouvons trouver les synonymes d'un mot ou d'un terme. Ceci aide à résoudre des problèmes de conflit de terme, qui se produisent quand les ontologistes conçoivent des ontologies en utilisant différemment des termes. Par exemple, l'un peut choisir le terme « `Personne` » pour le nom de la classe dénotant un individu, mais un autre peut décider d'employer le terme « `Humain` » à la place. Dans ces cas, l'utilisation des mesures de similarité basant sur la comparaison des chaînes de caractères telles que Jaro, Monge-Elkan… (*1.2.1.2.1*) pour déduire la similarité de deux classes à partir de leurs noms ne convient plus.

L'intégration de WordNet ou d'EuroWordNet dans les calculs de similarité entre deux noms de classes ou entre deux ensembles d'étiquettes de classe est effectuée en modifiant les mesures de similarité de nom (*Définition 20*) et des étiquettes (*Définition 21*).

Définition 25 *(Similarité des noms avec l'intégration de WordNet). Soit* n_1 *et* n_2 *deux noms de deux classes. Soit* N_1 *et* N_2 *ensembles des tokens obtenus après la normalisation de* n_1 *et de* n_2*, respectivement. Soit synset(t) fonction qui retourne l'ensemble de synonymes du terme t obtenu à partir de la consultation d'une bibliothèque telle que WordNet ou EuroWordNet. La similarité des noms avec l'intégration de WordNet est une fonction de la similarité* $S_{NomWN} : \mathcal{S} \times \mathcal{S} \rightarrow [0, 1]$ *telle que :*

$$S_{NomWN}(n_1, n_2) = \frac{\sum_{m_1 \in N_1} MJW(m_1, N_2') + \sum_{m_2 \in N_2} MJW(m_2, N_1')}{|N_1| + |N_2|}$$

$$\text{où } MJW(m_i, N) = \max_{m_j \in N} S_{Jaro-Winkler}(m_i, m_j)$$

$$N_i' = \bigcup_{n_k \in N_i} synset(n_k) \cup N_i$$

[1] http://www.illc.uva.nl/EuroWordNet/finalresults-ewn.html

Définition 26 *(Similarité des étiquettes avec l'intégration du WordNet). Soit $\mathcal{L}_1$ et $\mathcal{L}_2$ ensembles des étiquettes de deux classes de deux ontologies. Soit synset(t) fonction qui retourne l'ensemble de synonymes du terme t obtenu à partir de la consultation d'une bibliothèque de telle que WordNet ou EuroWordNet. La similarité des étiquettes avec l'intégration de WordNet est une fonction de la similarité $S_{ÉtiquetteWN} : 2^E \times 2^E \rightarrow [0, 1]$ telle que :*

$$S_{EtiquetteWN}(\mathcal{L}_1, \mathcal{L}_2) = \frac{\sum_{L_1 \in \mathcal{L}_1} MNS'(L_1, \mathcal{L}_2) + \sum_{L_2 \in \mathcal{L}_2} MNS'(L_2, \mathcal{L}_1)}{|\mathcal{L}_1| + |\mathcal{L}_2|}$$

$$\text{où } MNS'(L_i, \mathcal{L}) = \max_{L_j \in \mathcal{L}} S_{NomWN}(L_i, L_j)$$

$|\mathcal{L}_i|$, i = 1, 2, est la cardinalité de l'ensemble $\mathcal{L}_i$.

L'intégration de WordNet dans le calcul de la valeur de similarité entre des commentaires n'est pas utile et est coûteuse pour le temps du calcul car le commentaire est rarement un terme ou une expression courte. L'extension de longs commentaires textuels en ajoutant des synsets (ensembles des synonymes) pour chaque mot ou terme dans ces commentaires donne comme résultat plus de bruit que des informations utiles. Les commentaires contiennent eux-mêmes relativement assez d'information descriptive pour pouvoir comparer aux autres en employant la technique TF/IDF.

2.3.2 La similarité au niveau de structure

La similarité structurelle de deux classes (de même pour deux relations) de deux ontologies est déduite de leurs rapports avec les autres classes dans ces ontologies. Les classes dans une ontologie en `RDF(S)` sont organisées dans une structure hiérarchique, avec des liens de subsomption (ou spécialisation) entre elles. L'algorithme ASCO1 exploite les rapports avec les classes voisines et les rapports avec les classes dans le chemin vers la racine de la hiérarchie pour calculer la similarité structurelle entre deux classes.

2.3.2.1 La similarité structurelle adjacente

L'algorithme ASCO1 repose sur l'hypothèse que si les *voisines* de deux classes sont similaires, ces deux classes sont aussi considérées comme similaires. Les voisines d'une classe sont les classes ayant un lien de subsomption (direct ou indirect) avec la classe en question. Nous considérons trois types de voisines : (i) les super-classes directes, qui sont les classes parent directes ayant un lien de subsomption vers la classe en question ; (ii) les sous-classes directes, qui sont les descendants ayant le lien de subsomption de la classe en question vers ces classes ; (iii) les classes sœurs, qui ont la même classe mère que celle de la classe en question.

Définition 27 *(Voisines d'une classe). Soit H_c hiérarchie de classes de bl'ontologie. Soit c une classe dans la hiérarchie H_c. Pred(c), Succ(c), Sibl(c) sont définis comme les ensembles des super-classes, des sous-classes, et des classes sœurs de la classe c dans la hiérarchie H_c, respectivement.*

Soit S_1, S_2 deux ensembles. Nous définissons SameSet(S_1, S_2) l'ensemble d'éléments dans S_1 qui sont similaires linguistiquement avec un élément quelconque dans S_2 (*2.3.1.4*) (*Algorithme 10*), et UnionSet(S_1, S_2) l'ensemble d'éléments dans S_1 auquel sont rajoutes les éléments de S_2 qui ne sont pas similaires linguistiquement avec un élément quelconque dans S_1 (*Algorithme 11*).

Définition 28 *(SameSet et UnionSet). Soit S_1 et S_2 deux ensembles. SameSet et UnionSet de ces deux ensembles sont définis comme suivant :*

$$SameSet(S_1,S_2)=\{s_i\in S_1 \mid \exists s_j\in S_2, l-similar(s_1,s_2)\}$$
$$UnionSet(S_1,S_2)=S_1\cup\{s_j\in S_2 \mid \forall s_i\in S_1, \neg l-similar(s_1,s_2)\}$$

Algorithme 9 Dans_Ensemble(entité, ensemble)

```
n <- nombre d'entité dans l'ensemble ensemble
Pour i de 1 à n
    Si entité et ensemble[i] sont linguistiquement similaires
        Retourner vrai
    Fin Si
Fin Pour

Retourner faux
```

Algorithme 10 Same_Set(ensemble1, ensemble2)

```
SS <- créer un ensemble vide

n1 <- nombre d'entité dans l'ensemble ensemble1
Pour i de 1 à n1
    Si Dans_Ensemble(ensemble1[i], ensemble2)
        SS <- Ajouter(ensemble1[i], SS)
    Fin Si
Fin Pour

Retourner SS
```

Algorithme 11 Union_Set(ensemble1, ensemble2)

```
US <- ensemble1

n2 <- nombre d'entité dans l'ensemble ensemble2
Pour i de 1 à n2
    Si not Dans_Ensemble(ensemble2[i], ensemble1)
        US <- Ajouter(ensemble2[i], US)
    Fin Si
```

Fin Pour

Retourner US

À partir des définitions de SameSet et de UnionSet, les définitions des ensembles des super-classes similaires, des sous-classes similaires, des classes sœurs similaires et aussi les définitions de l'union des super-classes, des sous-classes, des classes sœurs sont définies.

Définition 29 *(SamePred, UnionPred, SameSucc, UnionSucc, SameSibl, UnionSibl). Soit c_1 et c_2 deux classes de deux ontologies. Pred(c), Succ(c), Sibl(c) sont les ensembles des voisines de la classe c, définis dans Définition 27. SamePred, UnionPred, SameSucc, UnionSucc, SameSibl, UnionSibl sont les ensembles des classes définis comme suivant :*

$$SamePred(c_1,c_2)=SameSet(Pred(c_1),Pred(c_2))$$
$$UnionPred(c_1,c_2)=UnionSet(Pred(c_1),Pred(c_2))$$
$$SameSucc(c_1,c_2)=SameSet(Succ(c_1),Succ(c_2))$$
$$UnionSucc(c_1,c_2)=UnionSet(Succ(c_1),Succ(c_2))$$
$$SameSibl(c_1,c_2)=SameSet(Sibl(c_1),Sibl(c_2))$$
$$UnionSibl(c_1,c_2)=UnionSet(Sibl(c_1),Sibl(c_2))$$

Nous définissons P_{Pred}, P_{Succ}, P_{Sibl} comme les proportions de similarité des classes dans les ensembles Pred, Succ, Sibl de deux classes, respectivement.

Définition 30 *(Proportion de similarité). Soit c_1 et c_2 deux classes de deux ontologies. Les proportions de similarité $P_{Pred}(c_1,c_2)$, $P_{Succ}(c_1,c_2)$, $P_{Sibl}(c_1,c_2)$ entre les ensembles des voisines de ces deux classes sont définies comme suit :*

$$P_{Pred}(c_1,c_2)=\frac{|SamePred(c_1,c_2)|}{|UnionPred(c_1,c_2)|}$$
$$P_{Succ}(c_1,c_2)=\frac{|SameSucc(c_1,c_2)|}{|UnionSucc(c_1,c_2)|}$$
$$P_{Sibl}(c_1,c_2)=\frac{|SameSibl(c_1,c_2)|}{|UnionSibl(c_1,c_2)|}$$

Dans les cas où les deux classes de deux ontologies à comparer n'ont pas de voisines, la proportion de similarité correspondante est égale à 1. Par exemple, si les deux classes à comparer sont les feuilles dans les hiérarchies de classes, (c'est-à-dire, elles n'ont pas de descendants), la P_{Succ} entre elles a la valeur de 1 ; si elles n'ont pas de classes sœurs, P_{Sibl} est à 1 ; et si chaque classe à comparer est une des racines dans sa hiérarchie de classes, (c'est-à-dire, elles n'ont pas de super-classes), la proportion de similarité P_{Pred} entre elles est mise à la valeur de 1.

Enfin, la similarité structurelle adjacente (*Algorithme 12*) de deux classes est définie dans la *Définition 31*. Elle est la somme pondérée des proportions de similarité avec les poids prédéfinis. Les valeurs des poids sont choisies suivant la nature des ontologies à aligner. Par exemple, si les ontologies à aligner sont modélisées de façon qu'une classe peut être spécialisée en plusieurs sous-classes, on attribuera des valeurs élevées aux poids correspondants aux P_{Succ} et P_{Sibl}; et inversement, si la plupart des classes ont plusieurs super-classes, le poids correspondant à P_{Pred} aura une valeur plus élevée.

Définition 31 *(Similarité structurelle adjacente). Soit c_1 et c_2 deux classes de deux ontologies. La similarité structurelle adjacente de deux classes est une fonction de la similarité $S_{Voisine}$: $\mathcal{H}_c \times \mathcal{H}_c \rightarrow [0, 1]$ telle que :*

$$S_{Voisine}(c_1, c_2) = \sum_{k=1..3} P_k(c_1, c_2) * w_k$$

où $P_1 = P_{Pred}, P_2 = P_{Succ}, P_3 = P_{Sibl}$

$$\sum_{k=1..3} w_k = 1$$

Algorithme 12 Similarité_Structurelle_Adjacente(entité1, entité2)

```
Pred1 <- extraire des super-entités de l'entité entité1
Pred2 <- extraire des super-entités de l'entité entité2
Succ1 <- extraire des sous-entités de l'entité entité1
Succ2 <- extraire des sous-entités de l'entité entité2
Sibl1 <- extraire des entités sœurs de l'entité entité1
Sibl2 <- extraire des entités sœurs de l'entité entité2

Si |Union_Set(Pred1, Pred2)| = 0
    PPred <- 1
Sinon
    PPred <- |Same_Set(Pred1, Pred2)|/|Union_Set(Pred1, Pred2)|
Fin Si
Si |Union_Set(Succ1, Succ2)| = 0
    PSucc <- 1
Sinon
    PSucc <- |Same_Set(Succ1, Succ2)|/|Union_Set(Succ1, Succ2)|
Fin Si
Si |Union_Set(Sibl1, Sibl2)| = 0
    PSibl <- 1
Sinon
    PSibl <- |Same_Set(Sibl1, Sibl2)|/|Union_Set(Sibl1, Sibl2)|
Fin Si

résultat <- PPred * wPred + PSucc * wSucc + PSibl * wSibl
Retourner résultat
```

2.3.2.2 La similarité de chemin des classes

Les classes dans une ontologie ont des rapports de spécialisation entre elles. Si une classe est sous-classe d'une autre, elle spécialise cette dernière et hérite de toutes les caractéristiques de cette dernière. Dans cette hiérarchie de classes, toutes les classes (sauf les classes racines) sont descendantes d'une des classes racines, et une classe hérite toutes les caractéristiques de toutes les classes dans le chemin de la classe en question jusqu'à sa racine dans la hiérarchie. De cette observation, deux classes (de même pour deux relations) de deux ontologies sont considérées similaires si les classes dans deux chemins connectant les classes en question à leurs racines dans leurs hiérarchies de classes sont déjà similaires. À noter que dans le cas où il y a des multi-héritages, c'est-à-dire une entité a plusieurs super-entités (entités parentales), il y a plusieurs chemins d'une entité vers un ou plusieurs racines, le *chemin de la classe* se compose de tous les entités dans tous les chemins possibles partant de la classe en question vers un des racines de la hiérarchie.

> ***Définition 32*** *(Chemin des classes). Soit H_c la hiérarchie de classes de l'ontologie. Soit c une classe dans la hiérarchie H_c. Chemin(c) est défini comme l'ensemble de classes qui sont des classes dans tous les chemins possibles construits par les liens de subsomption, connectant la classe c à une des classes racines dans la hiérarchie H_c.*

La similarité entre deux chemins de deux classes (*Algorithme 13*) est définie dans la *Définition 33*. Dans le cas où toutes les deux classes à comparer sont les racines dans les hiérarchies, leurs chemins sont donc vides, leur similarité de chemin des classes S_{Chemin} est considérée égale à 1.

> ***Définition 33*** *(Similarité de chemin des classes). Soit c_1 et c_2 deux classes de deux ontologies. La similarité de chemin des classes S_{Chemin} entre deux classes c_1 et c_2 est définie comme suivant :*
>
> $$S_{\text{Chemin}}(c_1,c_2)=\frac{|SameChemin(c_1,c_2)|}{|UnionChemin(c_1,c_2)|}$$
> où $SameChemin(c_1,c_2)=SameSet(Chemin(c_1),Chemin(c_2))$
> $UnionChemin(c_1,c_2)=UnionSet(Chemin(c_1),Chemin(c_2))$

Algorithme 13 Similarité_Chemin_Des_Classes(entité1, entité2)

```
Chemin1 <- extraire des entités dans des chemins de l'entité
entité1
Chemin2 <- extraire des entités dans des chemins de l'entité
entité2

Si |Union_Set(Chemin1, Chemin2)| = 0
    résultat <- 1
Sinon
    nSS <- |Same_Set(Chemin1, Chemin2)|
```

```
    nUS <- |Union_Set(Chemin1, Chemin2)|
    résultat <- nSS / nUS
Fin Si

Retourner résultat
```

2.3.2.3 Combinaison des similarités structurelles

Comme la combinaison des similarités linguistiques, la combinaison des similarités structurelles est effectuée par l'agrégation de la similarité structurelle adjacente et de la similarité de chemin des classes en employant la technique de la somme pondérée (*Définition 18*). Nous affectons le poids associé à la similarité de chemin des classes, une valeur plus élevée si la structure hiérarchique de l'ontologie est profonde; une valeur plus petite lui est affectée s'il y a multi héritage ou plusieurs liens de subsomption entre des classes dans les deux ontologies.

> ***Définition 34*** *(Similarité structurelle de deux classes). Soit A et B deux classes de deux ontologies. La similarité structurelle de deux classes A et B est une fonction de la similarité* $S_{Structurelle}$ *:* $O \times O \rightarrow [0, 1]$ *telle que :*
>
> $$S_{Structure}(A,B) = S_{Voisine}(A,B) * w_{Voisine} + S_{Chemin}(A,B) * w_{Chemin}$$
> $$w_{Voisine} + w_{Chemin} = 1$$

2.3.3 La génération des correspondances

La sortie du processus d'alignement d'ontologies est une liste de correspondances entre deux ontologies d'entrée. Ce sont des correspondances entre les entités de même type des ontologies : entre une classe d'une ontologie et une autre classe de l'autre ontologie ; entre une relation d'une ontologie et une autre relation de l'autre ontologie ; entre deux instances de deux ontologies. Les correspondances peuvent être celles du type 1-1, où une entité d'une ontologie a une et seulement une entité correspondante dans l'autre ontologie ; ou 1-n : une entité d'une ontologie peut être correspondante à une ou plusieurs entités de l'autre ontologie.

L'algorithme ASCO1 retourne comme sortie les correspondances entre deux ontologies et leurs valeurs de similarité sous forme des triplets (e_1, e_2, *sim*) où e_1 est une entité (classe, relation ou instance) de la première ontologie, e_2 est une entité de même type que e_1 de la deuxième ontologie, et *sim* est la valeur de similarité entre ces deux entités. Deux entités dans un triplet sont considérées similaires l'une à l'autre, suivant la valeur de similarité *sim* entre elles. Cette valeur, appelée la *valeur de similarité finale*, est calculée, par la somme pondérée, à partir de la similarité linguistique et de la similarité structurelle présentées dans les sections *2.3.1* et *2.3.2* respectivement (*Algorithme 14*). Pour chaque entité de la première ontologie, ASCO1 cherche l'entité de la deuxième ontologie qui est la plus similaire (la valeur de similarité finale entre elles est la plus élevé). Si cette valeur de similarité finale dépasse un seuil de similarité prédéfini, ces deux entités sont ajoutées dans la liste des triplets (une paire d'entités et leur valeur de

similarité) à retourner par l'algorithme. ASCO1 donc retourne les correspondances du type 1-n.

Définition 35 *(Similarité finale). Soit A et B deux entités de même type de deux ontologies. La similarité finale de deux entités A et B est une fonction de la similarité* S_{Finale} *:* $O \times O \rightarrow [0, 1]$ *telle que :*

$$S_{Finale}(A,B) = S_{Linguistique}(A,B) * w_{Linguistique} + S_{Structure}(A,B) * w_{Structure}$$
$$w_{Linguistique} + w_{Structure} = 1$$

Algorithme 14 Similarité_Finale(entité1, entité2)

```
nom1 <- extraire du nom de l'entité entité1
nom2 <- extraire du nom de l'entité entité2
etiq1 <- extraire des étiquettes de l'entité entité1
etiq2 <- extraire des étiquettes de l'entité entité2

Sim_Nom <- Similarité_des_Noms(nom1, nom2)
Sim_Etiq <- Similarité_des_Étiquettes(etiq1, etiq2)
Sim_Cmt <- Similarité_des_Commentaires(entité1, entité2)

Sim_Adj <- Similarité_Structurelle_Adjacente(entité1, entité2)
Sim_Chemin <- Similarité_Chemin_Des_Classes(entité1, entité2)

Sim_Lin <- Sim_Nom * wNom + Sim_Etiq * wEtiq + Sim_Cmt * wCmt
Sim_Struc <- Sim_Adj * wAdj + Sim_Chemin * wChemin

résultat <- Sim_Lin * wLin + Sim_Struc * wStruc
Retourner résultat
```

Notons que les poids employés dans les formules de type de la somme pondérée (Définition 23, Définition 31, Définition 34, Définition 35) sont les nombres réels non négatifs prédéfinis et sont choisis en satisfaisant que leur somme soit toujours égale à 1. Cela assure que les mesures de similarité sont toujours normalisées.

Définition 36 *(Correspondance). Soit A et B deux entités de même type de deux ontologies. Soit* $S_{Similarité} \geq 0$ *un seuil de similarité prédéfini. Deux entités A et B sont considérées similaires, donc, à ajouter dans la liste des correspondances entre deux ontologies, si et seulement si :*

$$S_{Finale}(A,B) \geq S_{\text{Similarité}}$$

2.4 Conclusion

Dans ce chapitre, nous avons présenté l'algorithme nommé ASCO1 pour chercher des entités correspondantes entre deux ontologies représentées en `RDF(S)`. Ces entités correspondantes sont celles de même type : les correspondances entre des classes, entre les relations ou entre les instances de deux ontologies. L'algorithme proposé exploite des points forts du formalisme `RDF(S)` pour calculer la valeur de similarité entre deux entités de deux ontologies et ensuite déduire les correspondances grâce à une valeur du seuil de similarité. La capacité de représenter le nom, les étiquettes, les commentaires d'une classe, d'une relation ou d'une instance du `RDF(S)` via ses primitives `rdf:id`, `rdfs:label`, `rdfs:comment`, respectivement, est exploitée pour calculer la valeur de similarité linguistique entre deux entités. Les liens de subsomptions (spécialisations) entre des classes et entre des relations, représentés en `RDF(S)` par les primitives `rdfs:subClassOf` et `rdfs:subPropertyOf` respectivement, sont aussi exploités pour obtenir une valeur de similarité structurelle entre deux entités. Ces deux valeurs de similarité, linguistique et structurelle, sont combinées pour avoir une valeur de similarité finale et ensuite, déduire si deux entités en question sont similaires ou différentes.

L'algorithme ASCO1 est donc un des premiers algorithmes attaquant le problème d'alignement d'ontologies et spécialement pour les ontologies représentées en `RDF(S)`. Il emploie les mesures de similarité basant sur la comparaison des chaînes de caractères, la technique TF/IDF inspirée du domaine de la recherche d'information, l'intégration de WordNet pour produire la similarité linguistique entre deux entités. Il exploite aussi des heuristiques existantes dans la plupart des modélisations d'ontologie (la similarité des entités voisines, la similarité des entités dans le chemin vers la racine) pour le calcul de la similarité structurelle de deux entités. Ainsi, l'algorithme ASCO1 est une application et une intégration des mesures de similarité de base (voir *1.2.1*) en exploitant des caractéristiques du formalisme `RDF(S)` et diffère les autres approches dans le domaine de l'alignement des schémas ou des ontologies (voir *1.2.3*).

La position de ASCO1 en rapport avec les autres approches est montrée dans la *Figure 13*. Comme S-Match, ASCO1 procède une étape de traitement automatique du langage naturel pour des composantes textuelles des entités et ensuite les compare en employant des mesures basées sur la similarité des chaînes de caractères. Mais ASCO1 applique la mesure Jaro-Winkler qui retourne un résultat meilleur dans la plupart des cas et dans un temps de calcul plus réduit [Cohen *et al*., 2003]. ASCO1 n'exploite pas la similarité des instances pour déduire la similarité des classes ou des relations correspondantes à ces instances comme les approches GLUE ou NOM/QOM le font, bien que la similarité de deux instances est calculée analogiquement comme dans le calcul de similarité de deux classes ou de relations (grâce à leurs nom, leurs étiquettes et leurs commentaires). Au lieu de cela, ASCO1 déduit la similarité entre deux entités grâce aux similarités de leurs voisines et les autres entités dans les chemins vers les racines dans les hiérarchies. Dans ASCO1, l'agrégation des valeurs de similarité calculées par de différentes mesures à partir des informations extraites des ontologies est effectuée comme dans la plupart des approches, en employant la technique de la somme pondérée. Par

contre, ASCO1 est la seule approche qui exploite la technique de TF/IDF pour déduire la similarité de deux entités à partir de leurs commentaires textuels.

Pour chaque entité de la première ontologie, l'algorithme ASCO1 cherche l'entité de la deuxième ontologie ayant la valeur de similarité finale la plus élevée. Le résultat, les correspondances trouvées, n'est donc pas symétrique et est de type n-1 : une ou plusieurs entités de la première ontologies peuvent être correspondantes à une entité de deuxième ontologie et si nous échangeons les positions des ontologies, le résultat sera différent. Si la première ontologie se compose de n entités, la deuxième contient m entités, ASCO1 doit effectuer n * m calculs, donc est de la complexité de $O(n^2)$. Le calcul de la similarité finale se compose des calculs sur les noms, les étiquettes, les commentaires, les voisines et les entités dans les chemins vers les racines. Parmi ces calculs, le calcul de la similarité des commentaires, basé sur la technique TF/IDF, est le plus lourd. Il demande de construire l'univers des mots U et des vecteurs représentant les entités. Plus les entités des deux ontologies ont des commentaires, et plus ces commentaires sont longs, donc plus le nombre des mots dans l'univers U est élevé et plus les vecteurs sont longs, plus le calcul de la similarité des commentaires prend du temps (*Définition 22*). Les calculs des mesures basées sur les informations structurelles (voisines, chemins) dépendent les calculs des mesures linguistiques. Les voisines de deux entités sont considérées similaires si leur valeur de similarité linguistique dépasse un seuil (elles sont linguistiquement similaires). Ces valeurs de similarité linguistique sont stockées dans une matrice de similarité pour accélérer la vitesse de calcul.

Un des limites de la version actuelle de l'algorithme ASCO1 est qu'il n'exploite pas des informations de domaine ou de co-domaine des relations, représentées en `RDF(S)` par des primitives `rdfs:domain` et `rdfs:range`, respectivement. Ces informations sont utiles pour déduire la similarité de deux classes qui sont les domaines ou les co-domaines de deux relations, qui sont déjà similaires ou vice versa, déduire la similarité de deux relations à partir de la similarité des classes ayant ce type de rapport avec elles. L'information concernant le type d'une instance n'est pas exploitée non plus pour inférer la similarité. Ce type de cette information est contenu dans la primitive `rdf:type` en `RDF(S)`, pour exprimer qu'une instance appartient à une classe quelconque. L'algorithme ASCO1 se focalise sur les informations textuelles dans les descriptions des entités et les informations structurelles dans les hiérarchies de classes et de relations.

Ces travaux sur l'algorithme ASCO1 ont été publiés dans la conférence internationale sur des systèmes d'information d'entreprise, ICEIS 2004 à Porto, Portugal [Bach *et al.*, 2004].

Les relations entre l'algorithme ASCO1 et les autres approches d'alignement des ontologies avec les mesures de similarité utilisées (qui sont résumées dans la *Figure 5*, page 19) sont montrées dans la *Figure 13*. Nous voyons que ASCO1 exploite les informations structurelles (voisines, chemin), la mesure Jaro-Winkler, la technique TF/IDF… pour calculer les valeurs de similarité partielles et les combiner par la technique de la somme pondérée pour déduire la valeur de similarité finale de 2 entités.

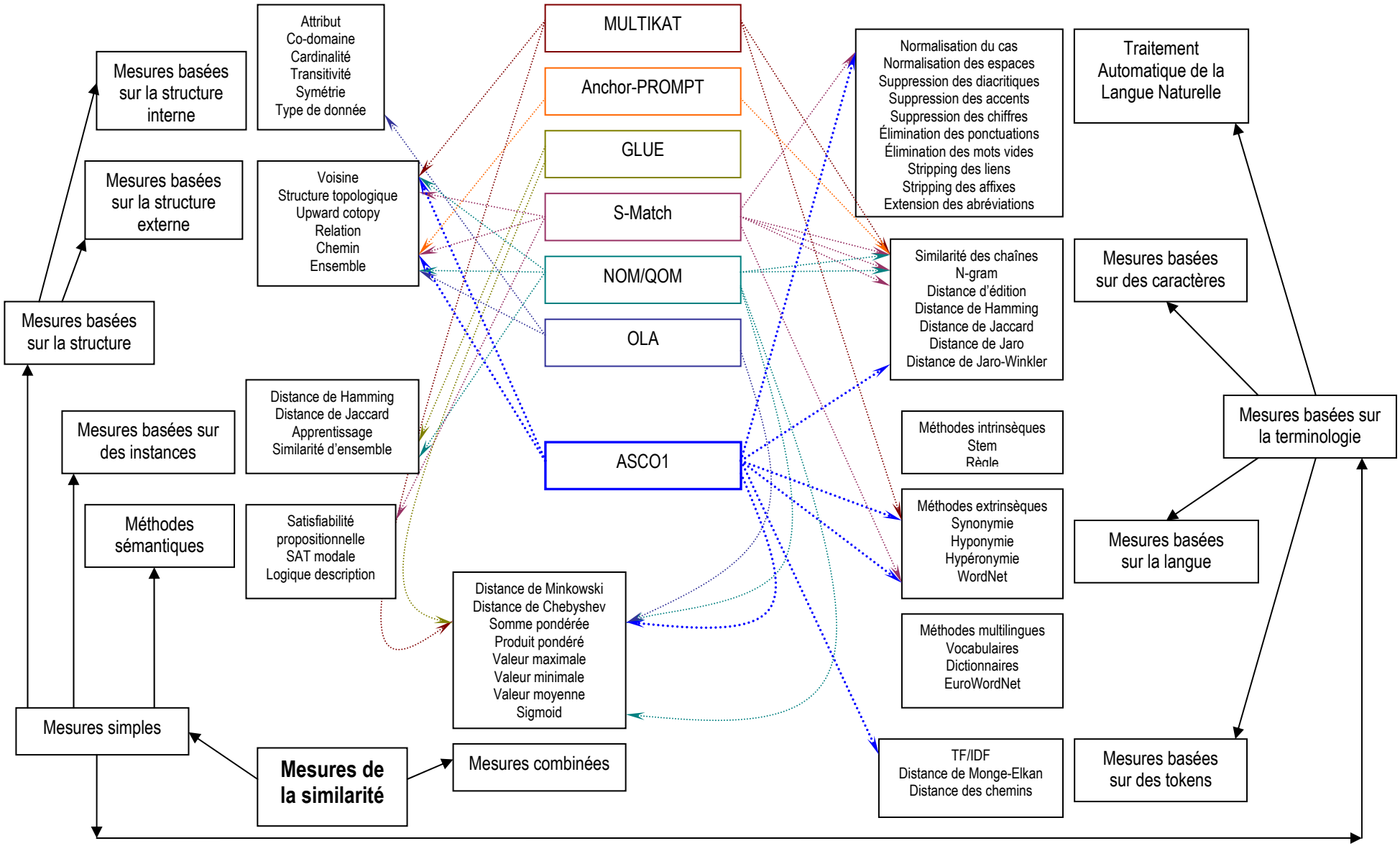


Figure 13 Le rapport entre l'algorithme ASCO1 et les autres approches d'alignement des ontologies

Chapitre 3

Algorithmes d'alignement pour ontologies en OWL

Dans ce chapitre, nous présentons deux algorithmes d'alignement d'ontologies représentées en langage d'ontologie recommandé par W3C : OWL. Du fait que le langage RDF(S) est moins expressif que le langage OWL, les deux algorithmes proposés sont conçus pour prendre en compte l'expressivité du langage OWL (avantage de OWL par rapport au langage RDF(S)). Le premier algorithme, nommé ASCO2, prend en compte principalement les informations sur la structure locale des entités des ontologies, contenues dans leurs descriptions, pour déduire la similarité entre les entités. Tandis que le deuxième algorithme, nommé ASCO3, analyse les rapports sémantiques entre les entités dans les ontologies (structure globale) pour extraire des correspondances entre des ontologies.

3.1 Alignement des Ontologies en OWL

L'algorithme d'alignement des ontologies représentées en RDF(S), ASCO1, présenté dans le chapitre 2, exploite le formalisme RDF(S) pour déduire la similarité entre deux entités de deux ontologies. RDF(S) ne permet que de représenter les ontologies simples ayant des classes, des relations et des instances avec leurs noms, leurs étiquettes et leurs commentaires. Certes que les classes et les relations dans les ontologies RDF(S) peuvent être organisées dans les hiérarchies (elles ont des super-classes ou super-relations) mais les ontologistes ne peuvent pas exprimer que deux relations sont équivalentes ou que deux classes sont disjointes. OWL[1] est un langage recommandé en 2004 par le World Wide Web Consortium W3C[2] pour représenter des ontologies dans le Web sémantique. Le langage OWL a un pouvoir d'expression plus fort que celui de RDF(S). Il permet d'exprimer non seulement la notion de l'intersection ou de l'union de plusieurs classes mais aussi les restrictions de valeur ou de cardinalité sur des propriétés pour certaines classes. Les relations dans une ontologie peuvent être aussi déclarées équivalentes en OWL ou certaines classes peuvent être déclarées disjointes mutuellement.

Bien que toutes les ontologies représentées en OWL soient aussi valides dans le formalisme RDF(S), c'est toujours intéressant que les algorithmes d'alignement des ontologies soient spécialement conçus pour les ontologies en OWL, au lieu de réutiliser ceux développés pour les ontologies en RDF(S). Cela permet d'exploiter des avantages et la capacité d'expression puissante du langage OWL en déduisant la similarité entre des entités des ontologies. L'algorithme ASCO1 peut être employé pour aligner des ontologies représentées en OWL, mais il n'emploie que des informations textuelles et descriptives des entités et il n'exploite pas les autres caractéristiques fournies par OWL.

Le langage OWL comporte trois sous-langages : OWL Lite, OWL DL, OWL Full, qui ont des capacités d'expression progressives, du simple au complexe. Chacun de ces sous-langages est une extension de son prédécesseur plus simple. Tout ce qui peut être légalement exprimé et valide dans OWL Lite et aussi légal et valide dans OWL DL ou bien OWL Full. Le sous-langage OWL Full a une expressivité maximum et une syntaxe libre (par exemple, il permet d'exprimer qu'une entité d'une ontologie est à la fois une instance et une classe), et donc il n'assure pas la complétude computationnelle (toutes les conclusions sont garanties pour être calculées) et la décidabilité. Il est improbable qu'il existe actuellement des logiciels permettant des raisonnements complets dans OWL Full, et à notre connaissance il n'existe pas d'ontologies en OWL Full. Ainsi, nous étudions des algorithmes qui cherchent des correspondances entre deux ontologies représentées en OWL DL. Ces algorithmes sont donc aussi applicables dans la tâche d'alignement des ontologies représentées en OWL Lite.

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

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

Dans ce chapitre, nous présentons deux algorithmes d'alignement d'ontologies conçus pour chercher des correspondances entre deux ontologies représentées en OWL DL (et aussi en OWL Lite).

3.2 Algorithme d'alignement ASCO2

Une entité d'une ontologie représentée en OWL DL est une classe, une relation ou une instance (un individu). Toutes les entités dans les ontologies en OWL DL sont des ressources (la notion dans le formalisme du RDF) et peuvent être identifiées par des URIs. Les classes et les instances peuvent être nommées ou pas. Dans le premier cas, elles sont référencées par les URIs et leurs noms sont les URIs qui les réfèrent. Les URIs se composent en deux parties : la partie de l'espace de noms (*namespace* en anglais) et le nom local. Dans le deuxième cas, ces classes ou ces instances sont appelées les nœuds vides (blank nodes) et elles ne sont pas référencées par aucun URI. Les ontologies OWL DL sont des documents valides en RDF(S). Les entités dans l'ontologie OWL DL sont décrites en employant des constructeurs (des primitives) du langage OWL. Ces descriptions sont interprétables sous forme des triplets RDF : (sujet, prédicat, objet), où les entités à décrire sont les sujets des triplets, les prédicats des triplets sont des primitives de OWL, et les objets sont les ressources (par exemple les entités, les littéraux…)

Code	Propriété de OWL	Domaine	Co-domaine	N
1	rdf:ID	-	-	1
2	rdf:type	rdfs:Resource	rdfs:Class	≥1
3	rdfs:label	rdfs:Resource	rdfs:Literal	≥1
4	rdfs:comment	rdfs:Resource	rdfs:Literal	≥1
5	rdfs:domain	rdf:Property	rdfs:Class	≥1
6	rdfs:range	rdf:Property	rdfs:Class	≥1
7	rdfs:subClassOf	rdfs:Class	rdfs:Class	≥1
8	rdfs:subPropertyOf	rdf:Property	rdf:Property	≥1
9	owl:equivalentClass	owl:Class	owl:Class	≥1
10	owl:equivalentProperty	rdf:Property	rdf:Property	≥1
11	owl:sameAs	owl:Thing	owl:Thing	≥1
12	owl:complementOf	owl:Class	owl:Class	1
13	owl:inverseOf	owl:ObjectProperty	owl:ObjectProperty	1
14	owl:differentFrom	owl:Thing	owl:Thing	≥1
15	owl:disjointWith	owl:Class	owl:Class	≥1
16	owl:onProperty	owl:Restriction	rdf:Property	1
17	owl:allValuesFrom	owl:Restriction	owl:Class	1
18	owl:someValuesFrom	owl:Restriction	owl:Class	1
19	owl:hasValue	owl:Restriction	-	1
20	owl:cardinality	owl:Restriction	xsd:nonNegativeInteger OWL Lite : {0, 1} OWL DL/Full : {0..N}	1
21	owl:minCardinality	owl:Restriction	xsd:nonNegativeInteger OWL Lite : {0, 1} OWL DL/Full : {0..N}	1
22	owl:maxCardinality	owl:Restriction	xsd:nonNegativeInteger OWL Lite : {0, 1} OWL DL/Full : {0..N}	1
23	owl:distinctMembers	owl:AllDifferent	rdf:List	1
24	owl:intersectionOf	owl:Class	rdf:List	1
25	owl:unionOf	owl:Class	rdf:List	1
26	owl:oneOf	owl:Class	rdf:List	1
27	rdfs:seeAlso	rdfs:Resource	rdfs:Resource	≥1

28	rdfs:isDefinedBy	rdfs:Resource	rdfs:Resource	≥1
29	owl:versionInfo	-	-	1
30	owl:priorVersion	owl:Ontology	owl:Ontology	≥1
31	owl:incompatibleWith	owl:Ontology	owl:Ontology	≥1
32	owl:backwardCompatibleWith	owl:Ontology	owl:Ontology	≥1
33	owl:imports	owl:Ontology	owl:Ontology	≥1

Tableau 4 Les propriétés de RDF(S) et de OWL, avec leurs domaines et co-domaines. N est le nombre de fois que la propriété peut apparaître dans la description d'une entité

La description d'une classe ou d'une relation dans une ontologie `OWL` est donc réalisée par un ensemble de triplets RDF, dont le sujet correspond à la classe ou à la relation en question. Les prédicats dans les triplets de la description d'une entité sont des primitives de `OWL`, qui sont les propriétés du langage `OWL` et celles du langage `RDF` (voir le *Tableau 4*). Il y a 33 propriétés avec leurs sémantiques prédéfinies dans `OWL` et `RDF(S)`, elles sont employées dans les descriptions de l'entité pour fournir la définition de l'entité. Chaque propriété utilisée dans un triplet apporte une pièce de connaissance à propos de l'entité à décrire. La combinaison de toutes ces pièces construit une définition de l'entité.

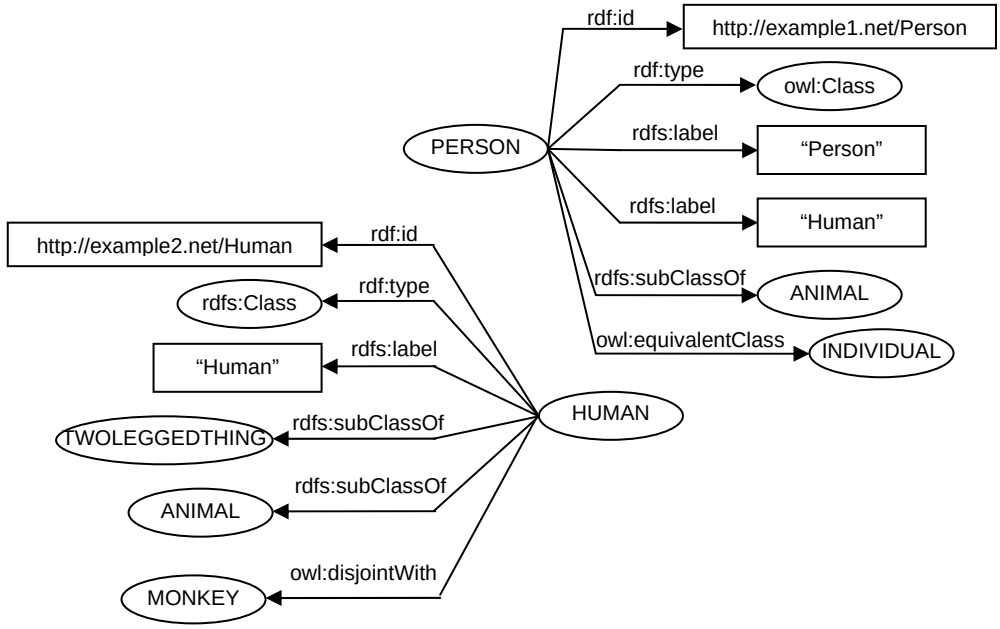


Figure 14 Représentations de deux entités « HUMAN » et « PERSON » dans deux ontologies différentes

L'algorithme ASCO2 est proposé à partir des observations ci-dessus. La similarité entre deux entités des deux ontologies est déduite des similarités partielles entre les composantes correspondantes que entités en question. Ces composantes sont des pièces de connaissance contenues dans les définitions de l'entité en employant des primitives du langage `OWL`. Les valeurs de similarité partielle sont ensuite agrégées dans un schéma de pondération variable pour obtenir une meilleure valeur de similarité finale de deux entités.

3.2.1 Algorithme d'alignement ASCO2 en détail

Une ontologie en OWL DL est un document RDF valid et les descriptions des entités en OWL DL sont les ensembles des triplets RDF. Soit (s, p, o) un triplet. Une ontologie O en OWL DL est un ensemble de triplets RDF, noté $\mathcal{O} = \{ (s_i, p_i, o_i) \}$.

Une description d'une classe ou d'une relation de l'ontologie en OWL DL est un ensemble de triplets RDF, notée $E = \{ (s, p_i, o_i) \}$, $E \subseteq \mathcal{O}$, où s correspond à l'entité en question. Suivant la conceptualisation, l'ensemble de triplets d'une entité peut contenir un seul triplet ou plusieurs triplets. Il peut arriver que dans l'ensemble, il y ait plusieurs triplets ayant un même prédicat. Par exemple, OWL permet le multi-héritage, une classe peut être définie comme sous-classe de plusieurs classes. Dans ce cas, la description de la classe se compose de plusieurs triplets ayant le prédicat `rdfs:subClassOf`. Le nombre de fois qu'un certain prédicat peut exister dans la description d'une entité est indiqué dans le *Tableau 4*. Par exemple, dans la définition de la classe « homme », nous ne pouvons exprimer au maximum qu'une fois avec la primitive `owl:complementOf`, que la classe « homme » est complément de la classe « femme ». Par contre, nous pouvons décrire cette classe avec plusieurs étiquettes différentes.

La comparaison de similarité entre deux entités e_1 et e_2 correspond donc à la comparaison de deux ensembles de triplets $E_1 = \{ (s_1, p_{1i}, o_{1i}) \}$ et $E_2 = \{ (s_2, p_{2j}, o_{2j}) \}$ représentant les deux entités. La comparaison de deux ensembles de triplets dépend des prédicats p_{1i} et p_{2j}, des objets o_{1i} et o_{2j}, et est effectuée comme suit :

Pour chaque prédicat p, nous obtenons deux ensembles $O_1 = \{ o_{1i} \mid (s_1, p, o_{1i}) \}$ et $O_2 = \{ o_{2j} \mid (s_2, p, o_{2j}) \}$.

La similarité entre deux ensembles O_1 et O_2 est calculée, notée $S_{Ensemble}(O_1, O_2)$. Nous appelons cette similarité la *similarité partielle* de deux entités sur le prédicat p, noté $S_p(e_1, e_2)$. $S_p(e_1, e_2) = S_{Ensemble}(O_1, O_2)$.

La similarité finale de deux entités e_1 et e_2, $S_{Finale}(e_1, e_2)$, est calculée en agrégeant les valeurs de similarité partielle obtenue de l'étape 2 pour chaque p_*. L'agrégation est réalisée dans un *schéma de pondération variable*.

La similarité finale S_{Finale} est stockée dans la matrice de similarité M_{Sim} (qui contient toutes les valeurs de similarité finales de deux entités de deux ontologies). Les valeurs de similarité dans cette matrice sont réutilisées pour calculer la similarité de deux autres entités. L'algorithme ASCO2 donc effectue un calcul de point fixe pour mettre à jour les valeurs de similarité entre deux entités après chaque itération (*Algorithme 15*). L'itération se terminer après un certain nombre d'itérations ou quand les valeurs de similarité dans la matrice M_{Sim} deviennent stables (la différence entre les valeurs de deux itérations consécutives inférieure à seuil prédéfini).

Algorithme 15 ASCO2(ontologie1, ontologie2)

```
MSim <- 0 // initialiser la matrice de similarité
```

```
Pour chaque entité e1 de l'ontologie ontologie1
  Pour chaque entité e2 de l'ontologie ontologie2
    MSim[e1, e2] <- Sim_Ling(e1, e2)
  Fin Pour
Fin Pour

Tant que le nombre d'itérations est inférieur à un seuil Ti et
         il y a encore des changements dans MSim
  Pour chaque entité e1 de l'ontologie ontologie1
    Pour chaque entité e2 de l'ontologie ontologie2
      MSim[e1, e2] <- Sim_Finale(e1, e2, MSim)
    Fin Pour
  Fin Pour
Fin Tant que

Retourner MSim
```

La similarité de deux ensembles dans l'étape 2 est calculée à partir de leurs éléments (*Définition 37*). L'étape 1 assure que les éléments dans ces ensembles sont de même type et donc ils sont comparables. Le type de ces éléments sont le co-domaine du prédicat. Les éléments peuvent être des littéraux tels que les chaînes de caractères, les nombres (`rdfs:Literal`), les classes (`rdfs:Class`), les propriétés (`rdf:Property`, ou `owl:ObjectProperty`), les instances (`owl:Thing`), les ontologies (`owl:Ontology`) (voir le *Tableau 4*).

> ***Définition 37*** *(Similarité des ensembles). Soit O_1 et O_2 deux ensembles d'éléments de même type. La similarité des ensembles de deux ensembles est une fonction de la similarité $S_{Ensemble} : 2^E \times 2^E \rightarrow \mathbb{R}$ telle que :*
>
> $$S_{Ensemble}(O_1, O_2) = \frac{\sum_{o_1 \in O_1} MSim(o_1, O_2) + \sum_{o_2 \in O_2} MSim(o_2, O_1)}{|O_1| + |O_2|}$$
>
> où $MSim(o_i, O) = \max_{o_j \in O} Sim(o_i, o_j)$

La matrice de similarité MSim est initialisée par les valeurs de similarité linguistique de deux entités calculées à partir de trois prédicats `rdf:id`, `rdfs:label`, `rdfs:comment` en employant des mesures de similarité comme dans l'algorithme ASCO1. L'*Algorithme 16* montre des étapes pour calculer la valeur de similarité linguistique de deux entités. Seuls les trois prédicats des triplets ayant des objets du type textuel (`rdfs:Literal`) sont pris en compte pour calculer les similarités partielles. Ces valeurs sont ensuite agrégées avec les poids prédéfinis pour obtenir la valeur de similarité linguistique finale.

Algorithme 16 Sim_Ling(entité1, entité2)

```
predicats1 <- l'ensemble de prédicats des triplets définissant
              l'entité entité1
predicats2 <- l'ensemble de prédicats des triplets définissant
              l'entité entité2
p_communs <- predicats1 ∩ predicats2
```

```
p_communs <- p_communs ∩ { p_id, p_label, p_comment }
Si p_communs est vide
  Retourner 0 // dans ce cas, la valeur de similarité linguistique
              // de deux entités entrées est égale à 0
              // c'est le cas des entités du type owl:Restriction
Fin Si

poids <- 0 // initialiser tous les poids à 0
poids[p_id] <- w_id // seulement les poids correspondants aux
poids[p_label] <- w_label // prédicats rdf:id, rdfs:label,
poids[p_comment] <- w_comment // rdfs:comment sont pris en compte

Somme <- 0
Pour chaque p ∈ p_communs
  Sim_Ling[p] <- calcul de la similarité linguistique de deux
                 entités avec le prédicat p (voir ASCO1)
  Somme <- Somme + Sim_Ling[p]* poids[p]
Fin Pour

Retourner Somme
```

3.2.1.1 Calcul des valeurs de similarité partielle

Rappelons que le prédicat p dans tous les triplets décrivant les entités classe ou relation appartient à l'ensemble de propriétés prédéfinies de `OWL` et de `RDF(S)` (voir le *Tableau 4*), c'est donc l'une de ces 33 propriétés. La similarité des ensembles (*Définition 37*) dépend du type du prédicat p. Nous examinons les différents types du p :

rdf:ID – Le triplet avec ce prédicat est utilisé pour identifier l'entité avec une URI. Nous appelons l'URI d'une entité son *nom*. L'URI se compose de deux parties : l'*espace de noms* et le *nom local*. La comparaison de deux noms de deux entités est effectuée sur leurs noms locaux. La description d'une entité ne contient au maximum qu'un seul triplet ayant `rdf:ID` comme son prédicat. Soit NomLocal(e) le nom local de l'entité e. La similarité des ensembles $S_{Ensemble}$ de deux entités e_1 et e_2 sur le prédicat `rdf:ID` devient $S_{Ensemble} = Sim_{ID}(e_1, e_2) = Sim(NomLocal(e_1), NomLocal(e_2))$. Cela est calculé en employant la normalisation du nom local et la mesure de similarité Jaro-Winkler, comme c'est le cas dans l'algorithme ASCO1 (*2.3.1.1*).

Code	Classe de OWL	Code	Classe de OWL
1	owl:AllDifferent	10	owl:Nothing
2	owl:AnnotationProperty	11	owl:ObjectProperty
3	owl:Class	12	owl:Ontology
4	owl:DataRange	13	owl:OntologyProperty
5	owl:DatatypeProperty	14	owl:Restriction
6	owl:DeprecatedClass	15	owl:SymmetricProperty
7	owl:DeprecatedProperty	16	owl:Thing
8	owl:FunctionalProperty	17	owl:TransitiveProperty
9	owl:InverseFunctionalProperty		

Tableau 5 Les classes de OWL

rdf:type – Toutes les entités dans une ontologie `OWL` valide sont typées par la primitive `rdf:type`. C'est-à-dire que pour chaque entité, il existe au moins un triplet dont le sujet est l'entité en question, le prédicat est `rdf:type`, et l'objet est une des classes de `OWL`. Les instances de l'ontologie sont typées par les classes définies (par les ontologistes) dans une ontologie. Les classes et les relations sont typées par les *classes prédéfinies de `OWL`*. Le langage `OWL` contient 17 classes prédéfinies avec des sémantiques bien précises (voir *Tableau 5*). Les classes de `OWL` sont organisées dans une taxonomie (*Figure 15*).

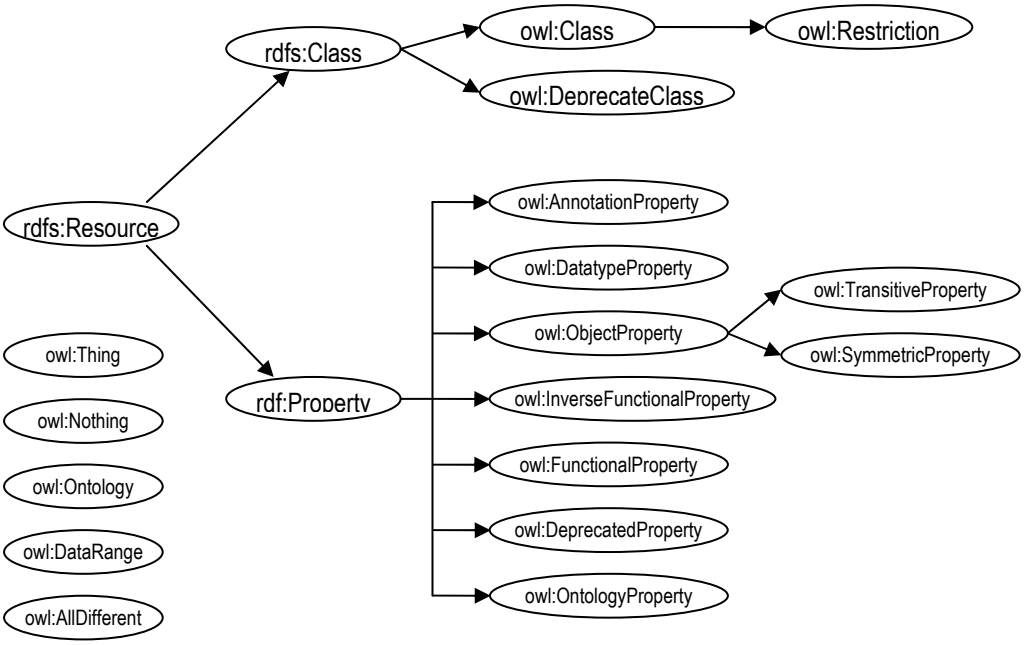


Figure 15 Taxonomie des classes de OWL

Les liens entre des classes `OWL` dans la taxonomie sont les liens de subsomption. Par exemple, la classe `owl:Class` est sous-classe de la classe `rdfs:Class` et aussi de la classe `rdfs:Resource`. Toutes les classes d'une ontologie en `OWL` (les entités définies avec le type de `owl:Class`) ont donc aussi le type `de rdfs:Resource`, c'est-à-dire que ce sont des ressources en `RDF(S)`. Cela nous permet de déduire la similarité entre deux entités à partir de leurs types (les classes de `OWL`). Par exemple, si dans une ontologie, la classe « `homme` » est typée avec la classe de `OWL owl:Class` et dans une autre ontologie, une autre classe « `homme` » est typée par `rdfs:Class`, la similarité de ces deux classes « `homme` » est déduite grâce au lien de subsomption entre `owl:Class` et `rdfs:Class`. Les similarités entre des classes de `OWL` sont prédéfinies et varient selon le domaine d'application de l'algorithme ASCO2. Le *Tableau 6* montre un extrait des paires des classes de `OWL` avec leurs valeurs de similarité. Ces valeurs de similarité sont symétriques. Par exemple, Sim(`owl:Restriction`, `owl:Class`) = Sim(`owl:Class`, `owl:Restriction`) = 0,4.

rdfs:label – La description d'une entité peut se composer de zéro, un ou plusieurs triplets ayant `rdfs:label` comme prédicat. La similarité de deux entités est déduite de

la similarité de deux ensembles $S_{Ensemble}$ (*Définition 37*) qui sont construits à partir des objets de ces prédicats. La similarité de deux objets, qui sont les étiquettes textuelles, $Sim(o_i, o_j)$ est calculée en employant la normalisation du nom local et la mesure de Jaro-Winkler, comme dans l'algorithme ASCO1 (*2.3.1.2*).

Classe de OWL	Classe de OWL	Valeur de similarité
owl:Restriction	owl:Restriction	1,0
owl:Restriction	owl:Class	0,4
owl:Restriction	owl:DeprecatedClass	0,1
owl:Restriction	rdfs:Class	0,2
owl:Restriction	rdfs:Resource	0
owl:Class	owl:Class	1,0
owl:Class	owl:DeprecatedClass	0,2
owl:Class	rdfs:Class	0,9
owl:Class	rdfs:Resource	0
…	…	…
owl:TransitiveProperty	owl:SymmetricProperty	0,3
owl:TransitiveProperty	owl:ObjectProperty	0,5
…	…	…

Tableau 6 Les valeurs de similarité prédéfinies entre les classes de OWL

rdfs:comment – Une entité peut être décrite avec zéro, un ou plusieurs commentaires en employant la primitive `rdfs:comment`. Dans le cas où elle a plusieurs commentaires (de même langue), ils sont agrégés pour faire la comparaison avec le commentaire agrégé d'une autre entité afin de déduire la similarité entre les deux entités. La comparaison est basée sur la technique de TF/IDF et est effectuée comme dans l'algorithme ASCO1 (*2.3.1.3*).

rdfs:domain, ***rdfs:range***, ***rdfs:subClassOf***, ***rdfs:subPropertyOf***, ***owl:differentFrom***, ***owl:disjointWith*** – Ce sont les primitives qui peuvent apparaître dans un ou plusieurs triplets décrivant une entité. La similarité de deux entités est déduite de la similarité des deux ensembles d'objets des triplets ayant ces primitives comme leurs prédicats. Les objets ont pour type une classe de `OWL`. Les valeurs de similarité entre ces objets (classes) $Sim(o_i, o_j)$ sont prises de la matrice de similarité M_{Sim}.

owl:equivalentClass, ***owl:equivalentProperty***, ***owl:sameAs*** – Ce sont les primitives qui peuvent apparaître dans un ou plusieurs triplets décrivant une entité. La similarité de deux entités est déduite de la similarité de deux objets les plus similaires dans les deux ensembles d'objets O_1 et O_2 (*Définition 38*). Les valeurs de similarité entre les objets de deux ensembles sont consultées dans la matrice de similarité M_{Sim}.

> ***Définition 38*** *(Similarité des ensembles, pour les prédicats owl:equivalentClass, owl:equivalentProperty, owl:sameAs). Soit O_1 et O_2 deux ensembles d'éléments qui sont les objets des triplets ayant soit le prédicat owl:equivalentClass, soit owl:equivalentProperty, ou soit owl:sameAs. La similarité de ces ensembles est une fonction de la similarité $S_{Ensemble} : 2^E \times 2^E \rightarrow \mathbb{R}$ telle que :*
>
> $$S_{Ensemble}(O_1, O_2) = \max_{o_i \in O_1, o_j \in O_2} M_{Sim}(o_i, o_j)$$

owl:complementOf, ***owl:inverseOf*** – Dans l'ensemble de triplets décrivant une entité, il y a au maximum un triplet ayant une primitive de ce type. La similarité de deux entités est déduite de la similarité de deux objets de deux triplets. La valeur de similarité de deux objets est prise de la matrice de similarité M_{Sim}. $S_{complementOf}(e_1, e_2) = M_{Sim}(o_1, o_2)$ et $S_{inverseOf}(e_1, e_2) = M_{Sim}(o_1, o_2)$.

owl:onProperty, ***owl:allValuesFrom***, ***owl:someValuesFrom*** – Ce sont des primitives qui n'apparaissent qu'au maximum une fois dans la définition d'une entité (qui est la classe anonyme du type `owl:Restriction`). La similarité de deux entités est donc déduite de la similarité des objets dans les triplets ayant ces primitives comme leurs prédicats. $S_{onProperty}(e_1, e_2) = M_{Sim}(o_1, o_2)$, $S_{allValueFrom}(e_1, e_2) = M_{Sim}(o_1, o_2)$, $S_{someValueFrom}(e_1, e_2) = M_{Sim}(o_1, o_2)$. La similarité $S_{ValueFrom}(e_1, e_2)$ est calculée dans le cas où e_1 et e_2 sont modélisées différemment en employant les primitives `owl:allValuesFrom` et `owl:someValuesFrom`. Examinons les deux descriptions d'exemple suivant :

```
<owl:Restriction>
    <owl:onProperty rdf:resource="#hasColor" />
    <owl:allValueFrom rdf:resource="#brightColors" />
</owl:Restriction>
```

et

```
<owl:Restriction>
    <owl:onProperty rdf:resource="#hasColor" />
    <owl:someValueFrom rdf:resource="#brightColors" />
</owl:Restriction>
```

Nous ne pouvons pas calculer la similarité $S_{allValueFrom}$ ou $S_{SomeValueFrom}$ pour deux classes du type de restriction `owl:Restriction` ci-dessus car elles ne possèdent pas les triplets ayant le même prédicat (à savoir `owl:allValueFrom` ou `owl:someValueFrom`). Mais il est intéressant d'exploiter la similarité $S_{ValueFrom}$ car dans ce cas, les descriptions ont quand même un lien. Cette connexion sera représentée par la similarité $S_{ValueFrom}$ entre deux entités.

owl:cardinality, ***owl:minCardinality***, ***owl:maxCardinality*** – Les $S_{cardinality}$, $S_{minCardinality}$, $S_{maxCardinality}$ sont égales à 1 si les objets des triplets (les nombres) sont exactement égaux l'un à l'autre. Dans le cas où nous pouvons déterminer la valeur maximale de l'espace de valeurs de l'objet (par exemple, la valeur de l'âge d'une personne ne dépasse pas 150 ans), le rapport de la différence des valeurs de la cardinalité (minimale, maximale) sur la valeur maximale est utilisée pour déduire la similarité de $S_{cardinality}$ ($S_{minCardinality}$, $S_{maxCardinality}$). Dans les autres cas, ces valeurs sont mises à zéro. Comme pour $S_{allValueFrom}$ et $S_{SomeValueFrom}$, la similarité S_{card} est calculée dans le cas où les descriptions des entités n'ont pas de triplets ayant le même prédicat de cardinalité (voir l'exemple suivant). S_{card} est calculée comme $S_{cardinality}$.

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasHusband" />
  <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">
    1
  </owl:cardinality>
</owl:Restriction>
```

et

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasHusband" />
  <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">
    1
  </owl:cardinality>
</owl:Restriction>
```

owl:distinctMembers, ***owl:intersectionOf***, ***owl:unionOf***, ***owl:oneOf*** – Les objets des triplets ayant ces primitives comme leurs prédicats sont du type `rdf:List`. L'algorithme ASCO2 effectue un traitement et considère donc les entités dans la liste dans un ensemble de triplets ayant le même prédicat et l'interprétation de cet ensemble est la combinaison des interprétations des triplets (l'opérateur AND). Voir l'exemple suivant :

```
<owl:Class rdf:ID="Person">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Man" />
    <owl:Class rdf:about="#Woman" />
  </owl:unionOf>
</owl:Class>
```

La description ci-dessus est traitée dans ASCO2 comme deux triplets (« `Person` », `owl:unionOf`, « `Man` ») et (« `Person` », `owl:unionOf`, « `Woman` »). Avec ce type de traitement, la similarité partielle sur la primitive `owl:unionOf` $S_{unionOf}$ de deux entités est calculée avec la formule dans la *Définition 37*.

Dans la version proposée, l'algorithme ASCO2 ne traite pas les primitives suivantes de `OWL` : `owl:versionInfo`, `owl:priorVersion`, `owl:incompatible-With`, `owl:backwardCompatibleWith` et `owl:imports` car ce sont les primitives supportant la gestion d'ontologie telle que la gestion des versions, de la compatibilité… et elles concernent le niveau de l'ontologie, mais pas le niveau des entités de l'ontologie (classe, relation, instance). Les primitives de `RDFS` qui peuvent être utilisées dans les ontologies en `OWL` : `rdfs:seeAlso`, `rdfs:isDefinedBy` ne sont pas prises en compte non plus en raison du typage des valeurs de ces primitives (`rdfs:Resource`). Il est difficile de comparer des objets non typés.

3.2.1.2 Agrégation des valeurs de similarité partielle

Rappelons que la similarité de deux entités de deux ontologies est déduite de leurs descriptions, qui sont les ensembles des triplets `RDF.` Les prédicats de ces triplets sont les primitives de `OWL` et de `RDF(S)` (voir *Tableau 4*). Pour chaque type de primitive, une valeur de similarité partielle est calculée pour deux entités en question : S_{ID}, S_{label}, $S_{comment}$, S_{domain}, S_{range}, $S_{subClassOf}$… Ces valeurs de similarité partielle dépendent des descriptions des entités, des triplets dans les ensembles, des primitives utilisées, des objets des triplets. La description d'une entité dépend de la conceptualisation et de la modélisation par l'ontologiste. Il peut employer certaines primitives autant de fois qu'il le souhaite pour définir une entité. Les triplets et le nombre des triplets dans la description d'une entité sont variés et non prédéterminés. L'application de la méthode de la somme pondérée avec les poids fixés pour agréger ces valeurs de similarité partielle semble donc inappropriée. L'algorithme ASCO2 emploie la méthode de la *somme pondérée avec les poids variables* pour produire la similarité finale de deux entités à partir des valeurs de similarité partielle (*Algorithme 17*).

Algorithme 17 Sim_Finale(entité1, entité2, MSim)

```
predicats1 <- l'ensemble de prédicats des triplets définissant
              l'entité entité1
predicats2 <- l'ensemble de prédicats des triplets définissant
              l'entité entité2
p_communs <- predicats1 ∩ predicats2
Si p_communs est vide
  Retourner 0 // dans ce cas, la valeur de similarité de deux
              // entités entrées est égale à 0
Fin Si

poids <- Calcul_des_Poids_Défaut(entité1, entité2, p_communs)
Somme <- 0
Pour chaque p ∈ p_communs
  Sim_Partielle[p] <- calcul de la similarité partielle de deux
                      entités avec le prédicat p et la matrice de
                      similarité MSim
  Somme <- Somme + Sim_Partielle[p]* poids[p]
Fin Pour

Retourner Somme
```

La méthode de la somme pondérée est employée dans plusieurs applications [Euzenat *et al.*, 2004] pour agréger des valeurs composantes dans une seule valeur. L'avantage de cette méthode est la possibilité de contrôler l'influence (ou l'importance) de chaque composante sur la valeur agrégée finale. Aux composantes les plus importantes seront associées des poids plus élevés, donc les valeurs de ces dimensions influenceront d'avantage la valeur agrégée finale. Les valeurs composantes sont les valeurs normalisées (entre l'intervalle de 0 à 1) et la somme des poids est égale à 1. Cela permet d'obtenir une valeur agrégée normalisée.

L'algorithme ASCO2 produit la valeur de similarité de deux entités à partir des valeurs de similarité partielle calculées pour chaque type de prédicat (il y a au total 33 types de prédicat différents - *Tableau 4*). Cependant, tous les types de prédicat ne peuvent être pas utilisés à la fois dans la définition d'une entité. En plus, suivant la modélisation, le même type d'entité peut être défini par certains prédicats plusieurs fois. Les valeurs de similarité partielle calculées sont très variées d'une entité à l'autre. Pour agréger ces valeurs partielles si variables, nous proposons une méthode appelée, la somme pondérée avec les poids variables, ou la *pondération variable*. L'idée de cette méthode s'inspire du mécanisme principal de la méthode d'origine, la somme pondérée avec les poids fixés, où chaque valeur composante est associée à un poids prédéfini et fixé, suivant son importance et la somme de ces poids est égale à 1. Dans notre méthode, ces poids peuvent varier suivant la nature des valeurs composantes. Du fait que tous les prédicats ne soient pas employés pour définir une entité, nous ne pouvons pas calculer toutes les valeurs de similarité partielle de deux entités. Dans ce cas, les poids prédéfinis associés aux valeurs de similarité partielle qui sont absentes, sont remis à zéro. Les valeurs de ces poids sont rajoutées aux poids qui restent (les poids associés aux valeurs de similarité partielle correspondant aux prédicats présents dans la définition de l'entité). Ce mécanisme

d'agrégation assure une combinaison des valeurs de similarité partielle flexible, plus précise, et indépendante du type d'entité (classe, relation ou instance).

Prenons cet exemple simple suivant : supposons que nous n'avons que trois types de prédicat P_1, P_2 et P_3, et deux types d'entité T_1 et T_2. La définition de l'entité du type T_1 accepte tous les trois types de prédicat mais les entités du type T_2 ne peuvent être définies qu'avec les prédicats du type P_1 et P_2. Pour chaque type de prédicat, nous calculons une valeur de similarité partielle, S_{pi}, et nous l'associons à un poids de pondération w_i. La valeur de similarité finale de deux entités est l'agrégation des valeurs de similarité partielle en employant la méthode de la somme pondérée :

$$S_{Finale} = \sum_{i=1}^{3} S_{pi} * w_i$$

Pour les entités du type T_1, la valeur de similarité entre elles est calculée par cette méthode sans difficulté. Mais pour les entités du type T_2, comme elles ne peuvent pas être définies en employant le prédicat P_3, la similarité partielle S_{p3} est toujours égale à 0, et donc, la valeur de similarité finale maximale de deux entités du type T_2 est de $w_1 + w_2 < 1$ (si $w_3 > 0$), c'est-à-dire qu'il n'y a jamais deux entités du type T_2 qui sont exactement similaires (deux entités sont *exactement similaires* si leur valeur de similarité finale est égale à 1). Avec le mécanisme de modification automatique des poids de la méthode de pondération variable, suivant la nature des entités (leurs définitions), le poids w_3 est remis à zéro lors deux entités du type T_2 sont comparées. La valeur de w_3 est rajoutée pour le poids w_1 et w_2 (assurant que la somme de tous les poids soit toujours égale à 1). La valeur de similarité finale de deux entités du type T_2 peut atteindre la valeur de 1 (selon des valeurs de similarité partielle S_{p1} et S_{p2}).

En revanche, si nous considérons la valeur de similarité partielle S_{pi} calculée à partir des informations correspondant au prédicat P_i, est égale à 1 dans le cas où le prédicat P_i n'existe pas dans les définitions de deux entités (une manière habituelle), alors, dans l'exemple ci-dessus, la valeur de similarité totale des entités du type T_2 est toujours au moins de w_3. Si $w_3 > 0$, il n'y a jamais des entités du type T_2 qui sont totalement différentes (deux entités sont *totalement différentes* si leur valeur de similarité totale est égale à 0).

La modification des poids est effectuée selon différentes stratégies dépendant du domaine d'application de l'algorithme ASCO2. La stratégie par défaut dans ASCO2 est de distribuer de manière *égale* la valeur du poids remis à zéro à tous les poids restants (les poids associés aux prédicats existant dans la définition de l'entité). Pour chaque type de prédicat qui n'existe pas dans les définitions de deux entités à comparer, le poids correspondant est remis à zéro et sa valeur est rajoutée aux autres poids, qui correspondent aux prédicats existant dans les définitions des entités, appelés `p_communs` (voir l'*Algorithme 18*).

Algorithme 18 Calcul_des_Poids_Défaut(p_communs)

```
poids <- initialiser les poids prédéfinis correspondent à chaque
         prédicat
Pour chaque p parmi les 33 primitives de OWL/RDF(S)
  Si p ∉ p_communs
```

```
    Pour chaque q ∈ p_communs
      poids[q] <- poids[q] + poids[p] / |p_communs|
    Fin Pour
    poids[p] <- 0
  Fin Si
Fin Pour

Retourner poids
```

Une autre stratégie est de distribuer les poids selon les rapports des poids de début (prédéfinis) correspondant aux prédicats dans `p_communs` (voir l'*Algorithme 19*). Le but est de toujours assurer que (1) la somme totale de tous les poids après la redistribution est égale à 1, et (2) les rapports entre les poids modifiés correspondant aux prédicats dans `p_communs` sont inchangés, en comparaison avec les rapports entre ceux prédéfinis. Cette stratégie assure que l'importance de différents prédicats, représentée par le biais des poids pondérés, est toujours préservée des changements. Par exemple, au début de l'algorithme, nous spécifions que l'information obtenue du prédicat `rdf:label` est deux fois plus importante que celle de `owl:disjointWith`, via les poids pondérés de 0,2 et 0,1 respectivement, alors le rapport des nouveaux poids pondérés après la redistribution correspondant à ces prédicats est toujours de 2-1.

Algorithme 19 Calcul_des_Poids_Rapport(p_communs)

```
poids <- initialiser les poids prédéfinis correspondent à chaque
         prédicat
rapports <- initialiser les rapports des poids correspondant aux
            prédicats dans p_communs
p0 <- premier prédicat dans p_communs dont le poids n'est pas zéro
somme_rapports <- 0
Pour chaque p parmi les prédicats dans p_communs
  rapports[p] <- poids[p] / poids[p0]
  somme_rapports <- somme_rapports + rapports[p]
Fin Pour

poids[p0] <- 1 / somme_rapports
Pour chaque p parmi les prédicats dans p_communs
  poids[p] <- poids[p0] * rapports[p]
Fin Pour

Retourner poids
```

3.2.2 Conclusion

Nous avons proposé l'algorithme ASCO2 pour effectuer des calculs de similarité entre les entités (classes, propriétés) d'ontologies en `OWL DL` ou `OWL Lite`. À partir des valeurs de similarité calculées, les correspondances entre deux ontologies sont déduites si la valeur de similarité dépasse un seuil de similarité prédéfini. Le calcul de similarité de deux entités se base sur le principe : exploitation maximale de la similarité des descriptions en `OWL` des entités en question. Les définitions des entités sont considérées

comme des ensembles des triplets RDF, dont les prédicats sont les primitives prédéfinies de OWL et de RDF(S). La similarité de deux entités correspond donc à la similarité des deux ensembles de triplets décrivant les entités en question. Cette dernière est calculée en agrégeant des valeurs de similarité partielle calculées à partir de la similarité des ensembles des objets des triplets ayant le même prédicat. L'agrégation est effectuée selon une méthode proposée appelée la somme pondérée avec les poids variables. Cela permet d'avoir une agrégation plus flexible, efficace et précise.

Le calcul de la similarité de deux entités dépend donc des similarités des objets des triplets qui sont (dans la plupart des cas) aussi des entités. Le calcul est donc effectué d'une manière récursive. ASCO2 effectue alors le calcul du point fixe pour mettre à jour les valeurs de similarité après chaque itération. Une matrice de similarité est utilisée pour stocker les valeurs de similarité entre les entités. Ces valeurs sont mises à jour jusqu'à ce qu'il n'y ait plus des changements des valeurs dans la matrice ou jusqu'à ce qu'un certain nombre d'itération soit atteint.

L'algorithme ASCO2 exploite aussi la sémantique bien définie des primitives qui sont les propriétés de OWL pour calculer la similarité des ensembles des objets des triplets. Pour chaque type de prédicat (propriété de OWL), une mesure de similarité est conçue pour prendre en compte des aspects particuliers de la syntaxe et de la sémantique de ce prédicat. La sémantique des primitives owl:cardinality, owl:minCard-inality, owl:maxCardinality ou owl:allValueFrom, owl:someValueFrom est exploitée pour déduire la similarité de deux entités définies en employant ces primitives.

L'algorithme ASCO2 repose donc principalement sur la structure locale de l'entité pour déduire la similarité (*1.2.1.3.1*). Les informations linguistiques dans la définition d'une entité sont aussi exploitées comme dans l'algorithme ASCO1 avec les primitives rdf:id, rdfs:label et rdfs:comment. Les valeurs de similarité linguistique sont prises comme points de départ, pour propager la similarité d'une paire d'entités à une autre (dans le processus de mise à jour des valeurs de similarité dans la matrice de similarité, pendant les itérations). Du fait que dans le langage d'ontologie OWL, nous pouvons définir les propriétés (relations) en spécifiant leurs domaines et co-domaines (ce sont des classes), les valeurs de similarité des propriétés sont donc calculées à partir des valeurs de similarité des classes, mais pas dans le sens inverse car il y a aucun moyen en OWL permettant de définir des classes en employant les propriétés de OWL et en spécifiant des relations. Autrement dit, la valeur de similarité de deux classes peut être propagée vers les paires des autres classes (via la primitive rdfs:subClassOf) ou vers des paires des relations (via rdfs:domain, rdfs:range), mais la valeur de similarité de deux relations ne peut être propagée que vers des paires d'autres relations (via la primitive rdfs:subPropertyOf) mais vers pas des paires des classes. Cette limite est étudiée et enlevée dans l'algorithme ASCO3.

Les relations entre l'algorithme ASCO2 et les autres approches d'alignement des ontologies avec les mesures de similarité utilisées (qui sont résumées dans la *Figure 5*, page 19) sont montrées dans la *Figure 16*. Nous voyons que ASCO2 exploite certaines informations et mesures comme ASCO1 pour calculer les valeurs de similarité partielles, mais il les combine par la technique de la somme pondérée *variable* pour déduire la valeur de similarité finale de deux entités.

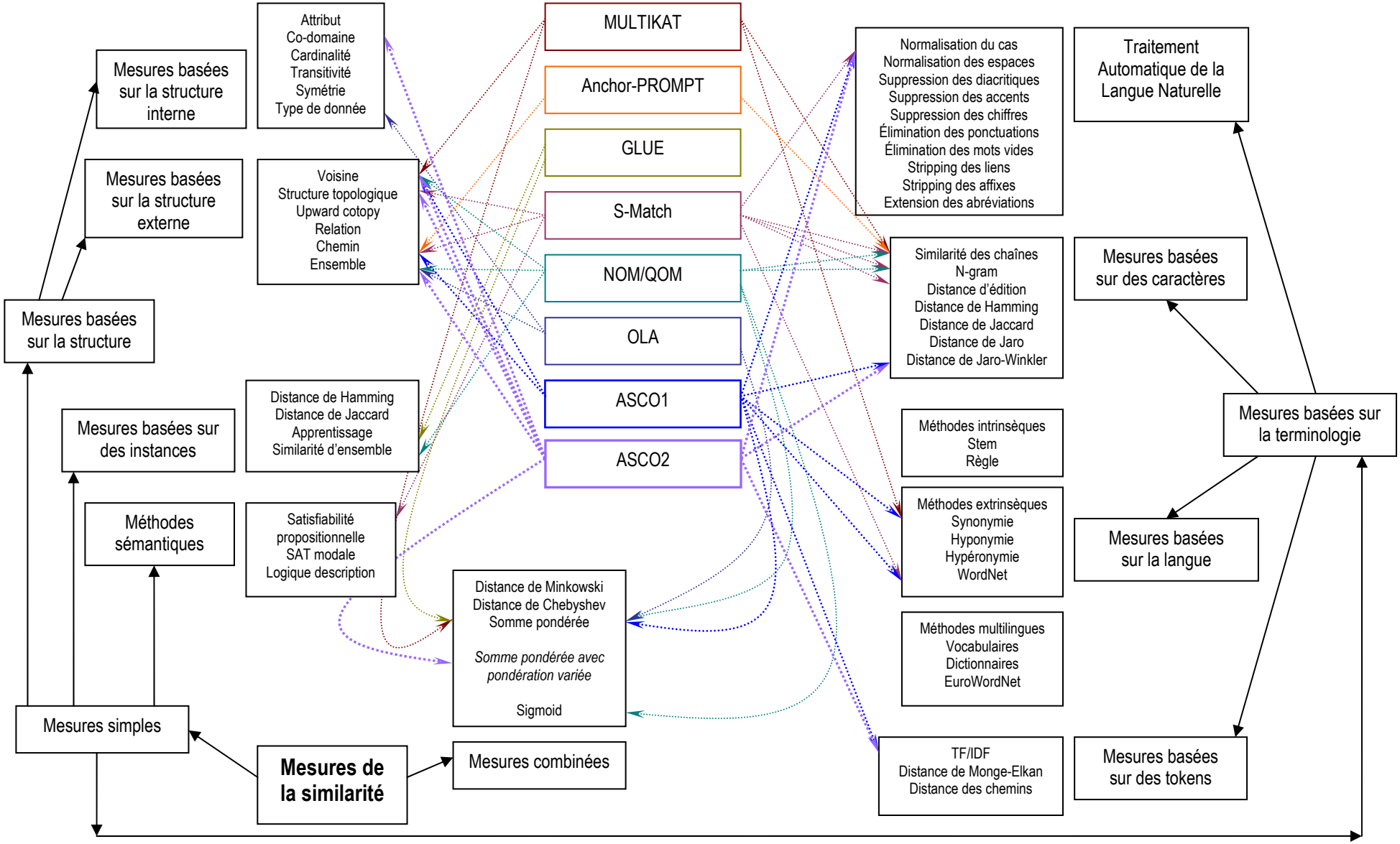


Figure 16 Le rapport entre l'algorithme ASCO2 et les autres approches d'alignement des ontologies

3.3 Algorithme d'alignement ASCO3

Dans cette section, nous présentons l'algorithme, nommé ASCO3, pour chercher des entités correspondantes entre deux ontologies représentées en OWL DL/Lite. Les limites de l'algorithme ASCO2, bien qu'il exploite quelques informations sémantiques des primitives du langage OWL, sont (i) qu'il déduit la similarité de deux entités en se basant principalement sur la structure locale (les descriptions) des entités, (ii) que la propagation de similarité (qui est en fait la déduction de similarité de deux entités à partir des autres entités ayant des relations avec les entités en question) est effectuée via quelques primitives (telles que rdfs:domain, rdfs:range, rdfs:subClassOf, rdfs:subPropertyOf) à sens unique : la similarité des relations est déduite de la similarité des classes ayant les liens de domaine et de co-domaine avec elles mais pas en sens inverse, la similarité des classes ne peut pas être déduite à partir de la similarité des relations ayant des liens avec les classes à comparer.

L'idée principale de l'algorithme ASCO3 est d'exploiter l'expressivité du langage d'ontologie OWL pour déduire la similarité entre des entités de deux ontologies. Les entités d'une ontologie sont décrites en employant les primitives de OWL (*Tableau 4*) avec leurs sémantiques bien définies. Nous pouvons alors considérer l'ontologie comme un *réseau sémantique* où les entités sont les nœuds dans le réseau et sont connectées par des liens qui sont les primitives de OWL. Ces liens ont des sémantiques spécifiées par les primitives. La connexion entre nœuds reflète la modélisation de l'ontologie dans le domaine pour lequel l'ontologie a été conçue et représentée. De cette observation, il s'en suit que si deux ontologies dans le même domaine sont similaires, leurs réseaux sémantiques sont aussi d'une certaine façon similaires. Ainsi, les nœuds (entités) se trouvant aux places correspondantes dans les parties similaires de deux réseaux sémantiques, sont considérés similaires. L'algorithme ASCO3 cherche donc les parties communes les plus grandes de deux réseaux sémantiques représentant deux ontologies, et ensuite décide si les entités aux positions correspondantes dans les parties trouvées sont similaires ou pas.

3.3.1 Algorithme d'alignement ASCO3 en détail

Le réseau sémantique correspondant à une ontologie est représenté de manière interne dans l'algorithme ASCO3 sous forme d'un graphe étiqueté, orienté et cyclique, appelé O-Graphe. Les entités de l'ontologie (classes, relations, instances) sont des nœuds du graphe, deux nœuds sont liés par un arc orienté et étiqueté par la primitive de OWL.

Définition 39 *(O-Graphe) Un O-Graphe est un 4-uplet og = (V, E, α, β), où*

- *V est l'ensemble fini de sommets (également appelés les nœuds)*
- *$E \subseteq V \times V$ est l'ensemble d'arcs*
- *$\alpha : V \rightarrow L$ est une fonction affectant une étiquette à chaque sommet*
- *$\beta : E \rightarrow L$ est une fonction affectant une étiquette (type) à chaque arc*

arc(u,v) est un arc orienté, commençant au nœud u et se terminant au nœud v. Deux fonctions permettent de récupérer les nœuds aux extrémités d'un arc :

n_gauche(arc(u,v)) = u et n_droit(arc(u,v)) = v. Nous notons n(e), le nœud dans le O-Graphe qui correspond à entité e de l'ontologie.

Rappelons qu'une ontologie OWL est un document RDF valide, et que les descriptions des entités en OWL sont interprétables sous forme des triplets RDF. Nous classons les propriétés (les primitives) de OWL en deux catégories : P_NE et P_Entité.

P_NE (*Tableau 7*) se compose des propriétés de OWL dont le type de co-domaine n'est pas une entité (classe, propriété et instance), telles que : rdfs:label, rdfs:comment (leur type de co-domaine est le littéral), owl:cardinality, owl:minCardinality, owl:maxCardinality (le type de leur co-domaine est l'ensemble des entiers possitifs), rdfs:seeAlso, rdfs:isDefinedBy (leur type de co-domaine est les ressources de RDF(S)), owl:priorVersion, owl:incompatibleWith, owl:backwardCompatibleWith, owl:imports (leur type de co-domaine est les ontologies OWL). P_NE inclut aussi rdf:ID, owl:hasValue et owl:versionInfo (leur type de co-domaine n'est pas précisé). Un triplet RDF dont le prédicat appartient à cette catégorie n'est pas pris en compte par l'algorithme ASCO3 pour construire le O-Graphe (les nœuds dans les O-Graphes sont des entités).

Code	Propriété de OWL	Domaine	Co-domaine
1	rdf:ID	-	-
2	rdfs:label	rdfs:Resource	rdfs:Literal
3	rdfs:comment	rdfs:Resource	rdfs:Literal
4	owl:cardinality	owl:Restriction	xsd:nonNegativeInteger OWL Lite : {0, 1} OWL DL/Full : {0..N}
5	owl:minCardinality	owl:Restriction	xsd:nonNegativeInteger OWL Lite : {0, 1} OWL DL/Full : {0..N}
6	owl:maxCardinality	owl:Restriction	xsd:nonNegativeInteger OWL Lite : {0, 1} OWL DL/Full : {0..N}
7	owl:hasValue	owl:Restriction	-
8	rdfs:seeAlso	rdfs:Resource	rdfs:Resource
9	rdfs:isDefinedBy	rdfs:Resource	rdfs:Resource
10	owl:versionInfo	-	-
11	owl:priorVersion	owl:Ontology	owl:Ontology
12	owl:incompatibleWith	owl:Ontology	owl:Ontology
13	owl:backwardCompatibleWith	owl:Ontology	owl:Ontology
14	owl:imports	owl:Ontology	owl:Ontology

Tableau 7 Les propriétés de OWL dans la catégorie P_NE

P_Entité se compose des 19 propriétés de OWL restantes (voir *Tableau 8*). Les descriptions des entités sous forme de triplets RDF ayant leur prédicat appartenant à la catégorie P_Entité sont exploitées pour construire le graphe O-Graphe.

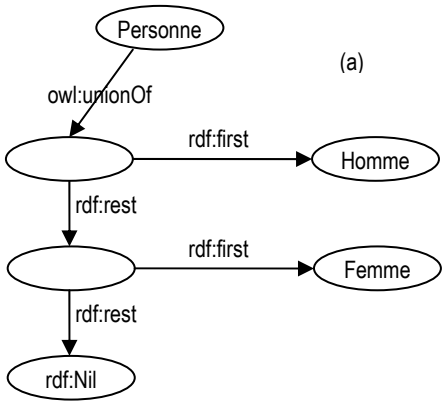
Code	Propriété de OWL	Domaine	Co-domaine
1	rdf:type	rdfs:Resource	rdfs:Class
2	rdfs:domain	rdf:Property	rdfs:Class
3	rdfs:range	rdf:Property	rdfs:Class
4	rdfs:subClassOf	rdfs:Class	rdfs:Class
5	rdfs:subPropertyOf	rdf:Property	rdf:Property
6	owl:equivalentClass	owl:Class	owl:Class
7	owl:equivalentProperty	rdf:Property	rdf:Property
8	owl:sameAs	owl:Thing	owl:Thing
9	owl:complementOf	owl:Class	owl:Class
10	owl:inverseOf	owl:ObjectProperty	owl:ObjectProperty

11	owl:differentFrom	owl:Thing	owl:Thing
12	owl:disjointWith	owl:Class	owl:Class
13	owl:onProperty	owl:Restriction	rdf:Property
14	owl:allValuesFrom	owl:Restriction	owl:Class
15	owl:someValuesFrom	owl:Restriction	owl:Class
16	owl:distinctMembers	owl:AllDifferent	rdf:List
17	owl:intersectionOf	owl:Class	rdf:List
18	owl:unionOf	owl:Class	rdf:List
19	owl:oneOf	owl:Class	rdf:List

Tableau 8 Les propriétés de OWL dans la catégorie P_Entité

Le graphe O-Graphe représentant l'ontologie en `OWL` à comparer est construit par l'*Algorithme 20*. L'entrée est l'ensemble de triplets `RDF` de l'ontologie. Pour chaque triplet, si le prédicat du triplet appartient à l'ensemble P_Entité, deux nœuds n_s et n_o correspondant au sujet et à l'objet du triplet, respectivement, seront ajoutés dans le graphe, s'ils n'existent pas encore, par la procédure Ajouter_Noeud. Un arc allant du nœud n_s au nœud n_o, avec étiquette correspondant au prédicat du triplet, est également ajouté dans le graphe. Si le prédicat du triplet est une des primitives `owl:equivalentClass`, `owl:equivalentProperty`, `owl:sameAs`, un arc inverse est créé entre le nœud objet et le nœud sujet, indiquant la même relation entre eux.

```
<owl:Class rdf:ID="Personne">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Homme" />
    <owl:Class rdf:about="#Femme" />
  </owl:unionOf>
</owl:Class>
```




```
<owl:Class rdf:ID="Personne">
  <owl:equivalentClass>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Homme" />
        <owl:Class rdf:about="#Femme" />
      </owl:unionOf>
    <owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

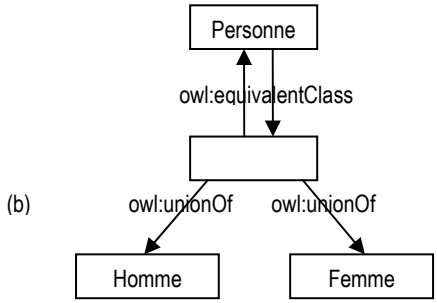


Figure 17 Exemple de deux définitions équivalentes de la classe `Personne` avec la primitive `owl:unionOf` et leurs représentations en Graphe RDF (a) et en O-Graphe (b)

Notons que, en `OWL`, le co-domaine des primitives `owl:distinctMembers`, `owl:intersectionOf`, `owl:unionOf`, `owl:oneOf` est une liste du type `rdf:List`. Comme dans l'algorithme ASCO2, ASCO3 traite les descriptions utilisant ces primitives comme les ensembles des triplets spéciaux, ayant la primitive comme le prédicat, et

l'objet dans la liste comme l'objet du triplet (voir *Figure 17*). La définition d'une classe employant une des primitives `owl:intersectionOf`, `owl:unionOf`, `owl:oneOf` est convertie en une définition équivalente utilisant une classe anonyme et des relations `owl:equivalentClass` avec la classe en question. Cela permet de préserver la signification de la définition avec des primitives `owl:intersectionOf`, `owl:unionOf`, `owl:oneOf` en utilisant une représentation plus facile à traiter. C'est-à-dire, la représentation en graphe `RDF` avec une liste des classes anonymes et des liens `rdf:first`, `rdf:rest` est transformée en la représentation plus simple en O-Graphe avec une seule classe anonyme et deux liens `owl:equivalentClass`. Les définitions utilisant la primitive `owl:distinctMembers` sont traitées par analogie, mais il n'est pas besoin de créer une ressource anonyme car le sujet des triplets ayant `owl:distinctMembers` comme le prédicat est déjà une ressource anonyme.

Algorithme 20 Construire_O-Graphe(ontologie)

```
O-Graphe <- initialiser le graphe

Pour chaque triplet t de l'ontologie ontologie
  Si prédicat(t) ∈ P_Entité
    Ajouter_Noeud(O-Graphe, sujet(t))
    Si prédicat(t) ∈ { owl:distinctMembers, owl:intersectionOf,
                       owl:unionOf, owl:oneOf }
      liste <- Prendre_Liste_Objets(t)
      nœud_gauche <- sujet(t)
      Si sujet(t) est une classe nommée
        res_anonyme <- créer une classe anonyme
        Ajouter_Noeud(O-Graphe, res_anonyme)
        Ajouter_Arc(O-Graphe, owl:equivalentClass,
                              sujet(t), nœud_gauche)
        Ajouter_Arc(O-Graphe, owl:equivalentClass,
                              nœud_gauche, sujet(t))
        nœud_gauche <- res_anonyme
      Fin Si
      Pour chaque objet o dans liste
        Ajouter_Noeud(O-Graphe, o)
        Ajouter_Arc(O-Graphe, prédicat(t), nœud_gauche, o)
      Fin Pour
    Sinon
      Ajouter_Noeud(O-Graphe, objet(t))
      Ajouter_Arc(O-Graphe, prédicat(t), sujet(t), objet(t))
    Fin Si
    Si prédicat(t) ∈ {owl:equivalentClass,owl:equivalentProperty,
                      owl:sameAs}
      Ajouter_Arc(O-Graphe, prédicat(t), objet(t), sujet(t))
    Fin Si
  Fin Si
Fin Pour

Retourner O-Graphe
```

La recherche des parties communes les plus grandes des deux réseaux sémantiques représentant les deux ontologies correspond donc à la recherche des plus grands sous-graphes communs de deux graphes O-Graphe. Ce problème appartient à la famille des problèmes de recherche de sous-graphe commun maximal. Les entités correspondantes de deux ontologies sont donc déduites des nœuds des deux graphes O-Graphe qui correspondent aux nœuds du sous-graphe commun maximal.

> ***Définition 40*** *(Sous-graphe commun et Sous-graphe commun maximal) Soit deux O-graphes* $og_1 = (V_1, E_1, \alpha_1, \beta_1)$ *et* $g_2 = (V_2, E_2, \alpha_2, \beta_2)$*. Un sous-graphe commun de* og_1 *et* og_2*, nommé* $ogc(g_1,g_2)$*, est un graphe* $og = (V, E, \alpha, \beta)$ *tel qu'il existe un isomorphisme de sous-graphe de og à* og_1 *et un isomorphisme de sous-graphe de og à* og_2*. Nous appelons og un sous-graphe commun maximal de* og_1 *et* og_2*, nommé* $ogcm(g_1, g_2)$*, s'il n'existe aucun autre sous-graphe commun de* og_1 *et* og_2 *qui ait plus de nœuds que og.*

Le problème de recherche de sous-graphe commun maximal est un problème NP-complet [Garey *et* Johnson, 1979]. Deux grandes approches étudient ce type de problème : l'approche de retour arrière (backtracking) [Levi, 1972], [McGregor, 1982] et l'approche de détection de la clique maximale [Durand *et al.,* 1999], [Bron *et* Kerbosch, 1973]. Ce sont toutes les deux des approches de recherche exhaustive (méthode de « *brute force* »). Certaines propositions emploient des heuristiques pour réduire la complexité de calcul, mais dans le pire cas, la complexité de ce type de problème est de O(N !) où N est le nombre des nœuds dans le graphe.

Les problèmes de recherche de sous-graphe commun maximal (SGCM) peuvent être transformés et résolus par le problème de détection de la clique maximale (CM) [Durand *et al.,* 1999]. Une *clique* est un graphe ayant ses nœuds (sommets) deux à deux adjacents (ils sont tous connectés l'un à l'autre). La première étape des algorithmes dans cette approche est la construction du graphe d'association à partir de deux graphes originaux. Les nœuds du graphe d'association correspondent aux paires de nœuds des deux graphes originaux. Les arcs du graphe d'association représentent la compatibilité des paires de nœuds (i.e. si des arcs de même type connectent les nœuds de deux paires dans deux graphes originaux ou pas). Le sous-graphe commun maximal est donc obtenu en cherchant la clique maximale dans le graphe d'association.

[Bunke *et al.,* 2002] montre dans les résultats des tests que la résolution du problème SGCM avec les graphes ayant une faible densité au niveau des nœuds et des arcs convient mieux pour l'approche de retour arrière, où l'on cherche tous les sous-graphes communs de deux graphes donnés et ensuite, l'on choisit le plus grand. Tandis que pour les graphes ayant une forte densité d'arcs, il est plus efficace de construire le graphe d'association à partir de deux graphes donnés, puis de rechercher la clique maximale de celui-ci. La raison est que dans le deuxième cas, où les graphes originaux ont beaucoup d'arcs, les informations de compatibilité des paires de nœuds des deux graphes (basées sur les arcs dans les graphes originaux) sont déjà encodées dans les arcs du graphe d'association, et cela accélère les calculs.

La compatibilité des paires de nœuds des deux O-Graphes est déduite à partir de plusieurs informations : le type de ces nœuds dans les O-Graphes (classe, propriété,

instance), les relations avec les nœuds voisins dans les O-Graphes (les types des arcs dans les O-Graphes), les voisins directs et indirects dans les O-Graphes (leurs types, leurs positions dans les O-Graphes). Pour accelérer la performance en général de l'algorithme ASCO3, les calculs de compatibilité des paires de nœuds sont effectués une seule fois et ces informations de compatibilité sont stockées dans le graphe d'association de deux O-Graphes. L'algorithme ASCO3 est donc conçu selon la deuxième approche : la recherche du sous-graphe commun maximal de deux graphes du type O-Graphe représentant deux ontologies est effectuée par la recherche de la clique maximale du graphe d'association de ces deux O-Graphes. La construction du graphe d'association prend en compte les différences entre les arcs des graphes O-Graphes et les sémantiques des arcs spécifiées par les primitives de `OWL`.

3.3.1.1 Construction du graphe d'association

Le but de la construction du graphe d'association est d'encoder les informations de compatibilité entre des nœuds des graphes O-Graphe dans une seule entité : l'arc du graphe d'association. Cela réduit le temps de calcul nécessaire chaque fois que l'algorithme devra chercher une paire des nœuds à ajouter dans le sous-graphe commun maximal (SGCM) et donc, cela accélère la performance de l'algorithme en général.

Généralement, le graphe d'association GA est construit à partir de deux graphes O-Graphe og_1 et og_2. Ses nœuds sont les combinaisons de deux nœuds : l'un du O-Graphe og_1, s_1, et l'autre du O-Graphe og_2, s_2, et noté $[s_1, s_2]$. Un arc est créé entre deux nœuds $[s_1, s_2]$ et $[t_1, t_2]$ si (1) la relation entre s_1 et t_1 dans og_1 et la relation entre s_2 et t_2 dans og_2 sont *compatibles* ou (2) s_1 et t_1 à la fois ne sont pas adjacents dans og_1 et s_2 et t_2 ne sont pas adjacents dans og_2.

> ***Définition 41*** *(Graphe d'association) Soit deux graphes de type O-Graphe* $og_1 = (V_1, E_1, \alpha_1, \beta_1)$ *et* $g_2 = (V_2, E_2, \alpha_2, \beta_2)$*. Le graphe d'association de* og_1 *et* og_2*, nommé* $ga(g_1,g_2)$*, est un O-Graphe* $ga = (V, E, \alpha, \beta)$ *où*
>
> - $V = \{[s_1, s_2] \mid s_1 \in V_1, s_2 \in V_2, s_1 \text{ et } s_2 \text{ sont n-compatibles}\}$
> - $E \subseteq V \times V$. $E = \{([s_1, s_2], [t_1, t_2]) \mid (s_1, t_1) \in E_1, (s_2, t_2) \in E_2, (s_1, t_1) \text{ et } (s_2, t_2) \text{ sont p-compatibles ; ou } (s_1, t_1) \notin E_1 \wedge (s_2, t_2) \notin E_2\}$
> - $\alpha : V \rightarrow L$ *est une fonction affectant une étiquette à chaque sommet*
> - $\beta : E \rightarrow L$ *est une fonction affectant une étiquette à chaque arc*

Nous allons définir la notion de « *compatibilité* » entre deux nœuds et deux arcs du graphe O-Graphe comme suit :

Deux nœuds s_1 du graphe og_1 et s_2 du graphe og_2 sont *n-compatibles* si les entités des ontologies représentées par ces nœuds sont le même type.

> ***Définition 42*** *(Compatibilité des nœuds) Soit deux nœuds* s_1 *et* s_2*. Soit* e_1 *et* e_2 *les entités de l'ontologie représentées par* s_1 *et* s_2*, respectivement.* s_1 *et* s_2 *sont n-compatibles si et seulement si :*
>
> - e_1 *et* e_2 *sont deux classes (leur type est soit rdfs:Class, soit owl:Class, soit owl:DeprecatedClass), ou*

- *e_1 et e_2 sont deux restrictions (leur type est owl:Restriction), ou*
- *e_1 et e_2 sont deux relations (leur type est sous-type de rdf:Property), ou*
- *e_1 et e_2 sont deux instances (leur type est une classe définie par utilisateur)*

Définition 43 *(Compatibilité des primitives) Soit deux primitives de OWL, p_1 et p_2. p_1 et p_2 sont p-compatibles si et seulement si :*

- *$p_1 = p_2$, ou*
- *$p_1, p_2 \in$ {owl:allValueFrom, owl:someValueFrom}, ou*
- *$p_1, p_2 \in$ {owl:cardinality, owl:minCardinality, owl:maxCardinality}*

La fonction p-comp(p_1, p_2) suivante est utilisée pour affecter une étiquette commune à un arc du graphe d'association, qui est créé à partir de deux triplets ayant p_1 et p_2 comme leurs prédicats (voir l'*Algorithme 21* – construction de graphe d'association).

Définition 44 *(p_comp) Soit deux primitives de OWL, p_1 et p_2. p_1 et p_2 sont p-compatibles. p_comp est une fonction renvoyant une primitive représentante pour p_1 et p_2 :*

- *p_comp(p_1, p_2) = p_1, si $p_1 = p_2$*
- *p_comp(p_1, p_2) = p_1, si $p_1, p_2 \in$ {owl:allValueFrom, owl:someValueFrom}*
- *p_comp(p_1, p_2) = p_1, si $p_1, p_2 \in$ {owl:cardinality, owl:minCardinality, owl:maxCardinality}*

Le graphe d'association est construit par l'*Algorithme 21*. En entrée, il prend deux O-Graphes construits grâce à l'*Algorithme 20*. Chaque paire de deux arcs, l'un du premier O-Graphe, l'autre du deuxième O-Graphe, est traitée pour construire deux nœuds et un arc du graphe d'association. Si les étiquettes des arcs (les primitives de `OWL`) sont p-compatibles (*Définition 43*), et les nœuds aux deux extrémités des arcs sont n-compatibles respectivement (*Définition 42*), alors un arc et deux nœuds sont créés dans le graphe d'association GA. Si le nœud à ajouter dans GA n'existe pas encore, `Ajouter_Noeud` va créer un nouveau nœud, sinon, il récupère le nœud existant.

L'algorithme ASCO3 applique la notion de « *saut* ». Il utilise la sémantique bien définie des primitives de `OWL` pour construire le graphe d'association. Le « *saut* » représente le cas où la compatibilité des nœuds peut être étendue via les liens sémantiques de `OWL`. Il est donc possible de déduire des correspondances des nœuds ayant les liens directs ou indirects. Par exemple, dans la *Figure 18*, nous présentons la représentation en O-Graphe de deux ontologies en `OWL`. La sémantique de la primitive de `OWL`, `owl:equivalentClass`, est prise en compte. Le graphe d'association GA comprend les deux arcs : [1,A]-P_0-[2,B] et [1,A]-P_0-[3,B] car le nœud 2 représente une classe équivalente à la classe représentée par le nœud 3 dans l'ontologie O_1. Notons que dans O_1, le nœud 1 est considéré comme adjacent avec le nœud 3, où il a un lien direct avec le nœud 3 (du fait de la primitive `owl:equivalentClass`) ; mais le nœud 1 et le nœud 4 ne sont pas adjacents. Il existe donc les arcs [1,A]-l_n-[4,C], [1,A]-l_n-[4,D], [1,A]-l_n-[5,C], [1,A]-l_n-[5,D] dans le graphe d'association GA. L'arc l_n dans GA indique une relation non-adjacente entre des nœuds dans le O-Graphe.

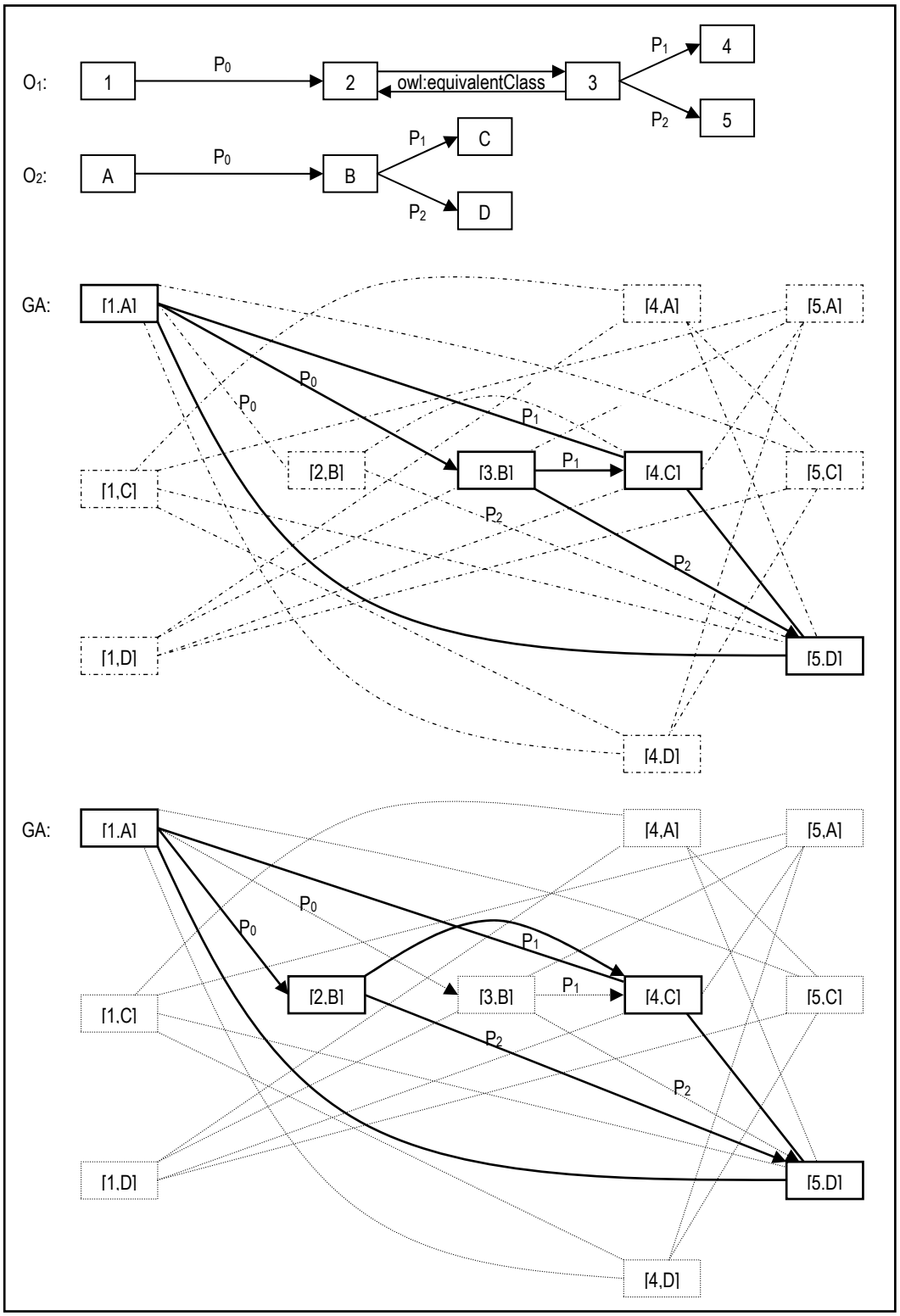


Figure 18 Exemple de deux ontologies simples, avec leurs représentations en O-Graphe, et leur graphe d'association, et deux cliques maximales

Avant d'introduire la notion de « *saut* » dans les différents cas avec de différentes primitives de `OWL`, nous définissons les notions de « *Parents d'une entité* », de « *Nœuds ancestraux* et *descendants de la i-ème génération* », de « *Nœuds ancestraux* et *descendants dans un rayon* » dans les définitions suivantes :

Définition 45 *(Parents d'une entité) Soit e une entité (une classe ou une propriété) de l'ontologie O. Les parents de l'entité e, Parents(e), sont les entités de l'ontologie O ayant le même type que e et ayant le lien de subsomption directe avec e :*

- *Parents(e) = { f | f est une classe, e est sous-classe directe de la classe f}, si e est une classe*
- *Parents(e) = { f | f est une propriété, e est sous-propriété directe de la propriété f}, si e est une propriété*

Autrement dit, `f` ∈ Parents(`e`) si et seulement s'il existe un triplet soit (`e rdfs:subClassOf f`) si `e` est une classe, soit (`e rdfs:subPropertyOf f`) si `e` est une propriété, dans l'ensemble de triplets définissant l'ontologie.

Définition 46 *(Nœuds ancestraux de la i-ème génération) Soit n(e) le nœud dans le O-Graphe og représentant une entité e (une classe ou une propriété) de l'ontologie O. Les nœuds ancestraux de la i-ème génération A^i du nœud n(e), $i \geq 0$, sont définies récursivement comme suit :*

- $A^i(n(e)) = \{ n(f) \mid f$ *est une entité de même type que e,* $\exists n(g) \in A^{i-1} \wedge f \in Parents(g) \}$
- $A^1(n(e)) = \{ n(f) \mid f \in Parents(e) \}$
- $A^0(n(e)) = \{ n(e) \}$

Les nœuds descendants de la i-ème génération sont définis par analogie.

Définition 47 *(Nœuds descendants de la i-ème génération) Soit n(e) le nœud dans le O-Graphe og représentant une entité e (une classe ou une propriété) de l'ontologie O. Les nœuds descendants de la i-ème génération D^i du nœud n(e), $i \geq 0$, sont définies récursivement comme suit :*

- $D^{i+1}(n(e)) = \{ n(f) \mid f$ *est une entité de même type que e,* $\exists n(g) \in D^i \wedge g \in Parents(f) \}$
- $D^1(n(e)) = \{ n(f) \mid e \in Parents(f) \}$
- $D^0(n(e)) = \{ n(e) \}$

Définition 48 *(Nœuds ancestraux dans un rayon) Soit n(e) le nœud dans le O-Graphe og représentant une entité e (une classe ou une propriété) de l'ontologie O. Les nœuds ancêtraux dans le rayon de r, A_r, $r \geq 0$, du nœud n(e) sont définies comme suit :*

- $A_r(n(e)) = \cup_{i=0..r} A^i(n(e))$

Les nœuds descendants dans un rayon, D_n, sont définis par analogie.

Définition 49 *(Nœuds descendants dans un rayon) Soit n(e) le nœud dans le O-Graphe og représentant une entité e (une classe ou une propriété) de l'ontologie O. Les nœuds descendants dans le rayon de r, D_r, $r \geq 0$, du nœud n(e) sont définies comme suit :*

- $D_r(n(e)) = \cup_{i=0..r} D^i(n(e))$

Le « *saut* » peut avoir lieu dans les cas suivants :

Premier cas : avec `rdf:type`.

Si deux nœuds qui sont des instances deviennent un nœud dans le graphe d'association GA, ce nœud sera considéré connecté directement par l'arc `rdf:type` aux nœuds représentant les classes de ces instances, ainsi qu'aux super-classes de ces classes (voir *Figure 19*).

Définition 50 *(Le saut avec la primitive rdf:type) Soit (i_1 rdf:type c_1) et (i_2 rdf:type c_2) deux triplets dans deux ensembles de triplets définissant deux ontologies O_1 et O_2, respectivement. Les nœuds et les arcs suivants sont ajoutés dans le graphe d'association GA de deux O-Graphes représentant deux ontologies O_1 et O_2, en prenant en compte la notion de « saut dans un rayon de r », avec la primitive rdf:type :*

- *Nœuds à ajouter dans GA :*
  - *$[n(i_1), n(i_2)]$*
  - *$[n(c_1), n2]$, $n2 \in A_r(n(c_2))$*
  - *$[n1, n(c_2)]$, $n1 \in A_r(n(c_1))$*
- *Arcs à ajouter dans GA :*
  - *$[n(i_1), n(i_2)]$-rdf:type-$[n(c_1), n2]$, $n2 \in A_r(n(c_2))$*
  - *$[n(i_1), n(i_2)]$-rdf:type-$[n1, n(c_2)]$, $n1 \in A_r(n(c_1))$*

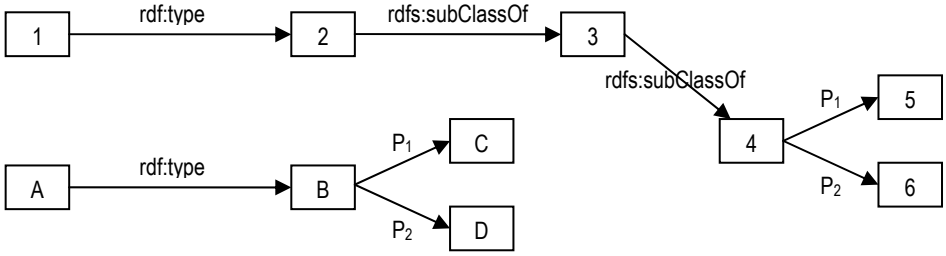


Figure 19 Le « *saut* » avec la primitive `rdf:type`

Dans cet exemple, pour le rayon de 0, nous avons un seul arc dans le graphe d'association : [1,A]-rdf:type-[2,B].

Pour le rayon de 1, nous avons deux arcs suivants dans le graphe d'association : [1,A]-rdf:type-[2,B] et [1,A]-rdf:type-[3,B].

Pour le rayon de 2, nous avons trois arcs suivants dans le graphe d'association : [1,A]-rdf:type-[2,B] , [1,A]-rdf:type-[3,B] et [1,A]-rdf:type-[4,B].

Le « *saut* » avec le rayon de 2 dans cet exemple permet de trouver une meilleure solution de mettre en correspondance des entités. Grâce à la sémantique bien définie de la primitive `rdfs:subClassOf`, si 1-A est une bonne correspondance, il est possible de déduire non seulement 2-B est une correspondance probable, mais aussi 3-B ou 4-B.

Deuxième cas : avec `rdfs:domain` et `rdfs:range`.

Le « *saut* » est possible pour les sous-classes de la classe qui est l'objet du triplet ayant `rdfs:domain` ou `rdfs:range` comme son prédicat (voir *Figure 20*).

Définition 51 *(Le saut avec les primitives rdfs:domain et rdfs:range) Soit p_{dr} une primitive de OWL, p_{dr} est soit rdfs:domain, soit rdfs:range. Soit (p_1 p_{dr} d_1) et (p_2 p_{dr} d_2) deux triplets dans deux ensembles de triplets définissant deux ontologies O_1 et O_2, respectivement. Les nœuds et les arcs suivants sont ajoutés dans le graphe d'association GA de deux O-Graphes représentant deux ontologies O_1 et O_2, en prenant en compte la notion de « saut dans un rayon de r », avec la primitive p_{dr} :*

- *Nœuds à ajouter dans GA :*
  - *$[n(p_1), n(p_2)]$*
  - *$[n(d_1), n2]$, $n2 \in D_r(n(d_2))$*
  - *$[n1, n(d_2)]$, $n1 \in D_r(n(d_1))$*
- *Arcs à ajouter dans GA :*
  - *$[n(p_1), n(p_2)]$- p_{dr} -$[n(d_1), n2]$, $n2 \in D_r(n(d_2))$*
  - *$[n(p_1), n(p_2)]$- p_{dr} -$[n1, n(d_2)]$, $n1 \in D_r(n(d_1))$*

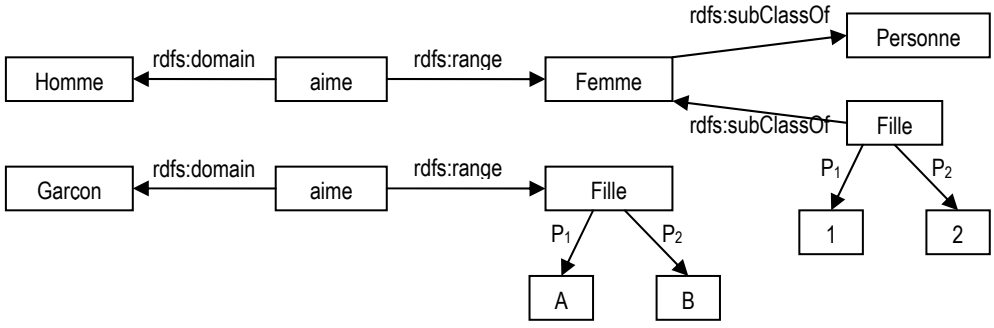


Figure 20 Le « *saut* » avec la primitive `rdfs:range`

Dans cet exemple, pour le rayon de *saut* descendant de 1, nous avons les arcs suivants dans le graphe d'association : [$aime_1$,$aime_2$]-rdfs:range-[$Femme_1$,$Fille_2$], [$aime_1$,$aime_2$]-rdfs:range-[$Fille_1$,$Fille_2$].

Notons qu'au contraire du *saut ancestral* dans le cas de la primitive `rdf:type`, où la déduction suivante est correcte : « si `i` est une instance de la classe `c`, `i` est aussi

une instance des classes d_k, d_k est une classe ancestrale de la classe c ($d_k \in A_r(n(c))$) », le *saut* pour les cas avec les primitives `rdfs:domain` et `rdfs:range` est le *saut descendant* : « si le domaine d'une propriété p est la classe c, les classes d_k, $d_k \in D_r(n(c))$, sont aussi domaine de cette propriété p ».

Troisième cas : avec `rdfs:subClassOf` et `rdfs:subPropertyOf`.

Le « *saut* » est possible pour les super-entités (super-classes, super-propriétés) de l'entité qui est l'objet du triplet ayant `rdfs:subClassOf` ou `rdfs:subPropertyOf` comme son prédicat (voir *Figure 21*).

Définition 52 *(Le saut avec les primitives rdfs:subClassOf et rdfs:subProperty-Of) Soit p_{cp} une primitive de OWL, p_{cp} est soit rdfs:subClassOf, soit rdfs:subPropertyOf. Soit (c_1 p_{cp} d_1) et (c_2 p_{cp} d_2) deux triplets dans deux ensembles de triplets définissant deux ontologies O_1 et O_2, respectivement. Les nœuds et les arcs suivants sont ajoutés dans le graphe d'association GA de deux O-Graphes représentant deux ontologies O_1 et O_2, en prenant en compte la notion de « saut dans un rayon de r », avec la primitive p_{cp} :*

- *Nœuds à ajouter dans GA :*
  - *$[n(c_1), n(c_2)]$*
  - *$[n(d_1), n2]$, $n2 \in A_r(n(d_2))$*
  - *$[n1, n(d_2)]$, $n1 \in A_r(n(d_1))$*
- *Arcs à ajouter dans GA :*
  - *$[n(c_1), n(c_2)]$- p_{cp} -$[n(d_1), n2]$, $n2 \in A_r(n(d_2))$*
  - *$[n(c_1), n(c_2)]$- p_{cp} -$[n1, n(d_2)]$, $n1 \in A_r(n(d_1))$*

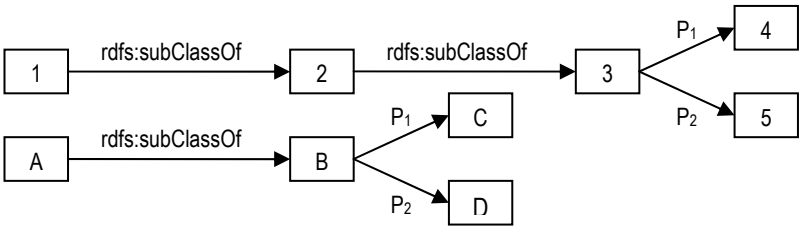


Figure 21 Le « *saut* » avec la primitive `rdfs:subClassOf`

Dans cet exemple, nous avons les arcs suivants dans le graphe d'association : [1,A]-rdf:subClassOf-[2,B], [1,A]-rdf:subClassOf-[3,B].

Quatriième cas : avec `owl:equivalentClass`, `owl:equivalentProperty` et `owl:sameAs`.

Pour toutes les triplets, le « *saut* » (sans rayon) est possible pour les entités équivalentes avec les entités qui sont le sujet et l'objet du triplet (voir *Figure 22*).

Définition 53 *(Équivalence des entités) Soit e et f deux entités (classe, propriété ou instance) de l'ontologie O. e et f sont e-équivalentes si et seulement si un des cas suivants est correct :*

- *e et f sont les instances et il existe un triplet soit (e owl:sameAs f), soit (f owl:sameAs e) dans l'ontologie O.*
- *e et f sont les classes et il existe un triplet soit (e owl:equivalentClass f), soit (f owl:equivalentClass e) dans l'ontologie O.*
- *e et f sont les propriétés et il existe un triplet soit (e owl:equivalentProperty f), soit (f owl:equivalentProperty e) dans l'ontologie O.*

Notons qu'une entité e est e-équivalente avec elle-même.

Définition 54 *(Nœuds équivalents) Soit n(e) le nœud dans le O-Graphe og représentant une entité e (une classe, une propriété ou une instance) de l'ontologie O. Les nœuds équivalents, NE, du nœud n(e) sont définies comme suit :*

- *NE(n(e)) = {n(f) | f est e-équivalente avec e}*

Notons que n(e) ∈ NE(n(e)).

Définition 55 *(Le saut avec l'équivalence des entités) Soit p_e une primitive de OWL. Soit (e_1 p_e f_1) et (e_2 p_e f_2) deux triplets dans deux ensembles de triplets définissant deux ontologies O_1 et O_2, respectivement. Les nœuds et les arcs suivants sont ajoutés dans le graphe d'association GA de deux O-Graphes représentant deux ontologies O_1 et O_2, en prenant en compte la notion de « saut», avec l'équivalence des entités :*

- *Nœuds à ajouter dans GA :*
  - *[n1, n2], n1 ∈ NE(n(e_1)), n2 ∈ NE(n(e_2))*
  - *[m1, m2], m1 ∈ NE(n(f_1)), m2 ∈ NE(n(f_2))*
- *Arcs à ajouter dans GA :*
  - *[n1, n2]- p_e -[m1, m2] , n1 ∈ NE(n(e_1)), n2 ∈ NE(n(e_2)), m1 ∈ NE(n(f_1)), m2 ∈ NE(n(f_2))*

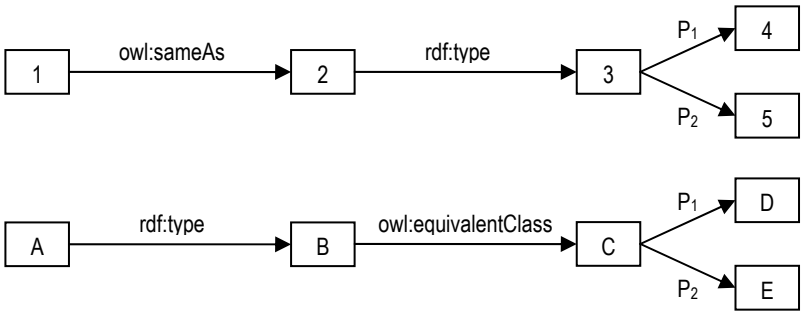


Figure 22 Le « *saut* » avec l'équivalence des entités

Dans cet exemple, nous avons les arcs suivants dans le graphe d'association : [1,A]-rdf:type-[3,B], [1,A]-rdf:type-[3,C], [2,A]-rdf:type-[3,B], [2,A]-rdf:type-[3,C].

Le « *saut* » est limité dans un rayon de `r` arcs, pour la raison de la sémantique du lien de subsomption. En général, plus `r` est élevé, plus la différence entre l'entité `e` et les entités dans la r-ème génération de `e` (A^r(e) ou D^r(e)) est importante. Il est donc nécessaire de limiter `r` pour que les significations des entités dans un rayon de `r` de l'entité `e` soient assez proches. La valeur raisonnable de `r` dépend de la modélisation des ontologies d'entrée, et devient deux paramètres d'entrée de l'*Algorithme 21* : `rSuper` et `rSous` pour le *saut ancestral* et le *saut descendant*, respectivement.

Notons que les arcs dans le graphe d'association représentent la compatibilité entre des nœuds dans les O-Graphes. Donc, ils sont des arcs non orientés.

Algorithme 21 Construction_Graphe_Association(og1, og2, rSuper, rSous)

```
GA <- initialiser le graphe d'association

Pour chaque arc arc1 dans le O-Graphe og1
  Pour chaque arc arc2 dans le O-Graphe og2
    Si étiquette(arc1) et étiquette(arc2) sont p-compatibles et
       n_gauche(arc1) et n_gauche(arc2) sont n-compatibles et
       n_droite(arc1) et n_droite(arc2) sont n-compatibles

      EE1_gauche <- équi_entités(n_gauche(arc1))
      EE1_droite <- équi_entités(n_droite(arc1))

      EE2_gauche <- équi_entités(n_gauche(arc2))
      EE2_droite <- équi_entités(n_droite(arc2))

      Pour chaque paire d'entités [e1g, e2g] dans
           EE1_gauche × EE2_gauche
        Ajouter_Noeud(GA, [e1g, e2g])
        Pour chaque paire d'entités [e1d, e2d] dans
             EE1_droite × EE2_droite
          Ajouter_Noeud(GA, [e1d, e2d])
          Ajouter_Arc(GA, p_comp(arc1, arc2),
                      [e1g, e2g], [e1d, e2d])
        Fin Pour
      Fin Pour

      Si (étiquette(arc1) est owl:subClassOf ou owl:subPropertyOf)
         ou (étiquette(arc1) est rdf:type et
             n_gauche(arc1) est une instance et
             n_gauche(arc2) est une instance)
        SEE1 <- super_et_équi_entités(n_droite(arc1), rSuper)
        Pour chaque paire d'entités [e1g, e2g] dans
             EE1_gauche × EE2_gauche
          Pour chaque paire d'entités [e1d, e2d] dans
               SEE1 × {n_droite(arc2)}
          Ajouter_Noeud(GA, [e1d, e2d])
          Ajouter_Arc(GA, p_comp(arc1, arc2),
                      [e1g, e2g], [e1d, e2d])
```

```
          Fin Pour
        Fin Pour

        SEE2 <- super_et_équi_classes(n_droite(arc2), rSuper)
        Pour chaque paire d'entités [e1g, e2g] dans
             EE1_gauche × EE2_gauche
          Pour chaque paire d'entités [e1d, e2d] dans
               {n_droite(arc1)} × SEE2
          Ajouter_Noeud(GA, [e1d, e2d])
          Ajouter_Arc(GA, p_comp(arc1, arc2),
                      [e1g, e2g], [e1d, e2d])
          Fin Pour
        Fin Pour
      Fin Si

      Si étiquette(arc1) est rdfs:domain ou rdfs:range
        // étiquette(arc2) doit exactement être étiquette(arc1)
        sEC1 <- sous_et_équi_classes(n_droite(arc1), rSous)
        Pour chaque paire d'entités [e1g, e2g] dans
             EE1_gauche × EE2_gauche
          // e1g, e2g sont des propriétés
          Pour chaque paire d'entités [e1d, e2d] dans
               sEC1 × {n_droite(arc2)}
          // e1d, e2d sont des classes
          Ajouter_Noeud(GA, [e1d, e2d])
          Ajouter_Arc(GA, p_comp(arc1, arc2),
                      [e1g, e2g], [e1d, e2d])
          Fin Pour
        Fin Pour

        sEC2 <- sous_et_équi_classes(n_droite(arc2), rSous)
        Pour chaque paire d'entités [e1g, e2g] dans
             EE1_gauche × EE2_gauche
          Pour chaque paire d'entités [e1d, e2d] dans
               {n_droite(arc1)} × sEC2
          Ajouter_Noeud(GA, [e1d, e2d])
          Ajouter_Arc(GA, p_comp(arc1, arc2),
                      [e1g, e2g], [e1d, e2d])
          Fin Pour
        Fin Pour
      Fin Si

    Fin Si
  Fin Pour
Fin Pour

// créer des arcs l_n du graphe d'association, qui indiquent une
// relation non-adjacent entre des nœuds dans O-Graphe
Pour chaque nœud n1 dans le O-Graphe og1
  NN1 <- l'ensemble de nœuds dans og1 qui ne sont pas
         adjacents à n1
  Pour chaque nœud n2 dans le O-Graphe og2
    NN2 <- l'ensemble de nœuds dans og2 qui ne sont pas
           adjacents à n2
```

```
    Si n1 et n2 sont n-compatibles
      Pour chaque nœud nn1 dans NN1
        Pour chaque nœud nn2 dans NN2
          Si nn1 et nn2 sont n-compatibles
            Ajouter_Noeud(GA, [n1, n2])
            Ajouter_Noeud(GA, [nn1, nn2])
            Ajouter_Arc(GA, l_n, [n1, n2], [nn1, nn2])
          Fin Si
        Fin Pour
      Fin Pour
    Fin Si

  Fin Pour
Fin Pour

Retourner GA
```

3.3.1.2 Recherche de la clique maximale dans le graphe d'association

Nous proposons un algorithme de recherche de clique maximale dans le graphe d'association (*Algorithme 22*). Cet algorithme est inspiré de l'algorithme proposé dans [Durand *et al.,* 1999]. La procédure `Recherche_Clique_Maximale` fonctionne de manière récursive. À chaque itération, l'algorithme essaie d'ajouter un nouveau sommet à `liste_sommets`, obtenue comme résultat dans l'itération précédente. La `liste_sommets` contient des sommets dans une clique et aussi des *sommets nuls* `SOMMET_NUL`. La taille de `liste_sommets` (nombre des sommets) détermine le niveau courant de calcul dans l'arbre de recherche.

Au début de l'algorithme, `liste_sommets` est initialisée comme une liste vide, elle ne contient aucun sommet. Cela correspond au résultat temporaire étant une clique vide, et au premier niveau de calcul, niveau 0, dans l'arbre de recherche.

Rappelons que nous cherchons la clique maximale dans le graphe d'association, construit à partir des nœuds (sommets) de deux graphes O-Graphe et que la clique maximale trouvée correspondra au sous-graphe commun maximal de ces deux graphes O-Graphe. Autrement dit, pour chaque sommet s_i du graphe og_1 qui appartient au sous-graphe commun maximal trouvé, il y a un et seulement un sommet s_j du graphe og_2 qui lui correspond et vice versa. Cela veut dire que dans la clique maximale trouvée du graphe d'association (la `liste_sommets`), le sommet $[s_i, s_j]$ existe au plus une fois (on ne peut jamais trouver le cas où $[s_i, s_{j1}]$ et $[s_i, s_{j2}]$ ou $[s_{i1}, s_j]$ et $[s_{i2}, s_j]$ co-existent dans la clique maximale). Alors, si les sommets $[x, y]$ dans le graphe d'association sont organisés dans des groupes selon x, et si à chaque niveau de recherche, nous ne considérons qu'un seul sommet dans le groupe correspondant à ce niveau, l'espace de recherche sera toujours assuré comme un arbre. Le niveau maximal, ou la profondeur, de l'arbre de recherche est le nombre de groupes, qui est au maximum le nombre de sommets de O-Graphe og_1.

La sortie de l'algorithme est une liste de sommets du graphe d'association `liste_sommets`, qui contient des sommets de la clique maximale trouvée. `liste_sommets` contient aussi des sommets nuls `SOMMET_NUL`. Le i-ème sommet dans `liste_sommets` correspond soit à un sommet du graphe d'association dans le i-ème groupe appartenant à la clique maximale trouvée, soit à un sommet nul. Le sommet nul `SOMMET_NUL` représente le cas où aucun sommet du graphe d'association dans le i-ème groupe n'appartient à la clique maximale trouvée.

Reprenons l'exemple dans la *Figure 18*, le graphe d'association GA de deux ontologies O_1 et O_2 se compose de 11 sommets sous forme de [x, y], où x et y sont deux sommets dans les O-Graphes représentant les ontologies O_1 et O_2, respectivement. Chaque sommet dans ces 11 sommets est connecté à au moins un autre sommet dans l'ensemble de ces 11 sommets. Les autres sommets qui sont des combinaisons possibles de deux ensembles de sommets de deux O-Graphes, tels que [2, A], [3, D]…, ne sont pas intéressants à être traités car ils sont des sommets isolés (ils ne sont pas connectés au aucun autre sommet) et donc, ils n'appartiendront jamais à une clique.

Les 11 sommets du graphe d'association sont organisés dans 5 groupes différents (*Tableau 9*), selon le premier élément qui est un sommet dans le O-Graphe og_1 :

Groupe	Sommets dans le groupe
G_1	[1, A] [1, C] [1, D]
G_2	[2, B]
G_3	[3, B]
G_4	[4, A] [4, C] [4, D]
G_5	[5, A] [5, C] [5, D]

Tableau 9 Groupes de sommets du graphe d'association

Dans cet exemple, le niveau maximal de l'arbre de recherche est égal au nombre de sommets dans le O-Graphe og_1 (égal à 5).

Si le niveau courant de la recherche atteint le niveau maximal `niveau_max`, qui est le nombre des nœuds dans le graphe og_1, nous arrivons à la feuille de l'arbre de recherche, le résultat courant (`liste_sommets`) est sauvegardé avec son meilleur statut (`nombre_sommets_nuls`). Plus le `nombre_sommets_nuls` est petit, plus la taille de la clique trouvée est grande. Ainsi, à chaque itération, si le `nombre_sommets_nuls` courant dépasse le `meilleur_nombre_sommets_nuls` obtenu jusqu'alors, l'algorithme ne continue plus, car l'ajout de nouveaux sommets dans la `liste_sommets` n'enlève jamais des sommets nuls qui y existent déjà.

Algorithme 22 Recherche_Clique_Maximale(liste_sommets)

```
niveau = taille(liste_sommets)
nombre_sommets_nuls = compter_sommets_nuls(liste_sommets)
clique_taille = niveau – nombre_sommets_nuls
Si nombre_sommets_nuls >= meilleur_nombre_sommets_nuls
  Retourner
Sinon
  Si niveau == niveau_max
    sauvegarder_resultat(liste_sommets)
    meilleur_nombre_sommets_nuls = nombre_sommets_nuls
  Sinon
    G = ensemble de sommets [s1,s2] ayant
        s1 correspond au niveau courant
    G = G ∪ { SOMMET_NUL }
    Pour chaque s dans G
      Si est_legal(s, liste_sommets)
        Recherche_Clique_Maximale(liste_sommets + s)
      Fin Si
    Fin Pour
  Fin Si
Fin Si
Retourner
```

Si le niveau maximal n'a pas encore été atteint, et si le nombre des sommets nuls dans la `liste_sommets` est toujours inférieur à `meilleur_nombre_sommets_nuls`, l'algorithme peut continuer à ajouter des nouveaux sommets pour élargir la clique courante. Alors, l'algorithme choisit successivement tous les sommets du graphe d'association dans le groupe `G` correspondant au niveau courant de recherche. Pour chaque sommet `s` dans `G`, la recherche est branchée si ce sommet `s` est un *sommet légal* (il est connecté à tous les sommets non nuls dans la `liste_sommets`, et donc l'ajout de ce sommet aboutit à une nouvelle clique plus grande que l'ancienne).

Notons qu'au groupe `G` est ajouté un sommet nul `SOMMET_NUL`. Le sommet nul est un sommet spécial. Il est considéré toujours légal : il est connecté à tous les autres sommets. La `liste_sommets` peut contenir plusieurs fois ce sommet spécial. Le `SOMMET_NUL` est ajouté dans le résultat courant `liste_sommets` lorsque l'algorithme a examiné tous les sommets dans `G`. Il s'agit alors du cas où le nœud du graphe og_1 correspondant au niveau courant de recherche s_1 n'a aucun candidat correspondant dans le graphe og_2. Autrement dit, le nœud s_1 de graphe og_1 n'appartient pas au graphe commun maximal de og_1 et og_2.

Toujours avec l'exemple dans la *Figure 18*, un extrait de l'arbre de recherche de clique est montré dans la *Figure 23*. Une des cliques maximales trouvées ayant la taille de 4 est dénotée par les liens en gras. Elle correspond à la `liste_sommets` = {[1,A], `SOMMET_NUL`, [3,B], [4,C], [5,D]}. Le deuxième sommet de cette liste est un sommet nul. Cela signifie qu'il n'y a aucun sommet de deuxième groupe G_2 appartenant à la clique maximale trouvée. Autrement dit, l'entité de la première ontologie représentée par le deuxième nœud de O-Graphe og_1 (2) n'a aucune entité correspondante dans la deuxième ontologie. Les correspondances entre deux ontologies sont donc `1-A`, `3-B`, `4-C` et `5-D`.

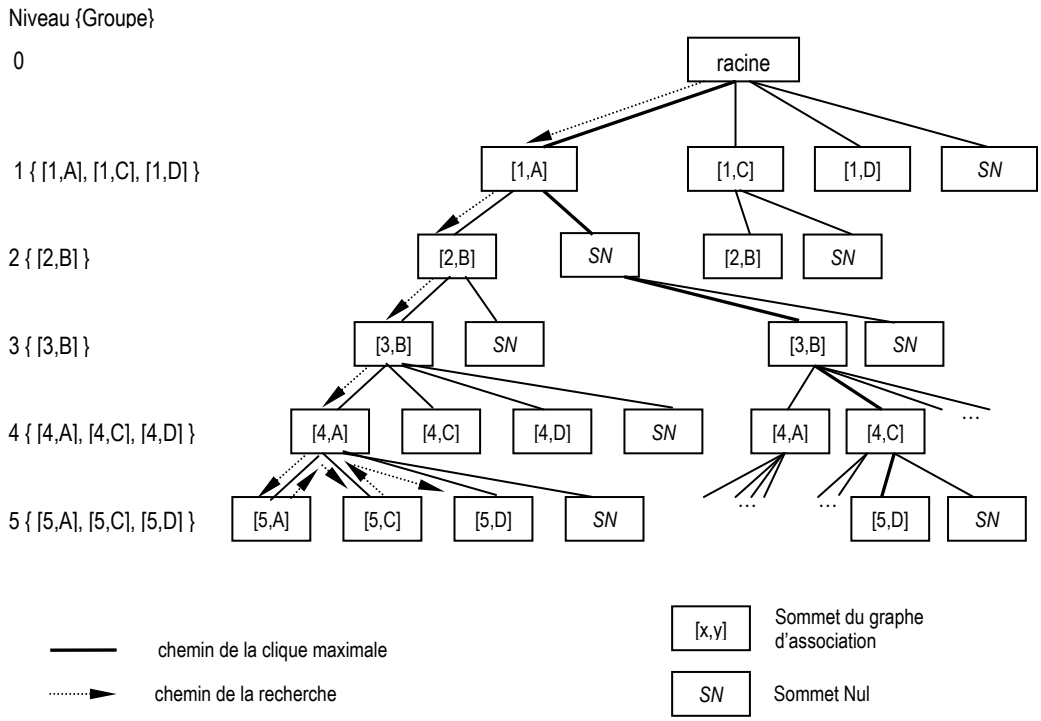


Figure 23 Extrait de l'arbre de recherche de clique

Notons qu'un graphe g = (V, E, α, β), avec V ≠ ∅, contient toujours au moins une clique. Si E = ∅, le graphe g n'a aucun arc, les cliques maximales de ce graphe sont les cliques ayant la taille de 1, chaque clique maximale se compose d'un seul nœud, et le nombre de cliques maximales est égal à |V|, le nombre de nœuds dans le graphe g. Si E ≠ ∅, la taille de la clique maximale est au moins de 2.

Il est possible d'avoir plusieurs cliques de même taille dans le graphe. Ainsi, il y a probablement plusieurs cliques ayant la même taille maximale. La sortie de l'*Algorithme 22* est un ensemble de cliques maximales de même taille, et chaque clique est une solution possible des correspondances entre deux ontologies. Le choix de la *meilleure clique maximale* est basé sur la similarité linguistique de deux entités (*Définition 23*) dans chaque nœud de la clique. Rappelons que chaque nœud du graphe d'association, donc de la clique, est construit à partir de deux nœuds du O-Graphe, et les nœuds dans les O-Graphes correspondent aux entités des ontologies. La similarité linguistique de deux entités dans un nœud de la clique est le poids pour ce nœud. Le poids total d'une clique est alors la somme de tous les poids de ses nœuds. Ainsi, la *meilleure clique maximale* dans l'ensemble de cliques maximales trouvées par l'*Algorithme 22* est celle qui a le poids total maximal.

La liste des correspondances entre deux ontologies est déduite à partir de la meilleure clique maximale en supprimant les nœuds nuls `SOMMET_NUL` qui sont ajoutés pendant la recherche de la clique. Chaque nœud de la meilleure clique maximale se compose de deux nœuds des O-Graphes, correspondant aux deux entités de deux ontologies. Elles sont alors considérés comme similaires.

3.3.1.3 Améliorations de l'algorithme

Le problème de recherche de clique maximale dans un graphe est un problème NP-complet [Garey *et* Johnson, 1979]. Si le graphe a `N` nœuds, la complexité est de O(`N` !) [Bunke *et al.,* 2002]. Dans notre problème, si les deux graphes O-Graphe ont `N` nœuds, le pire des cas est celui où les nœuds dans les O-Graphes sont reliés par un seul type d'arc (par exemple, seule la primitive `rdfs:subClassOf` est utilisée pour modéliser les ontologies), alors le graphe d'association est un graphe complet, avec $N \times N$ nœuds.

Examinons l'*Algorithme 22*, le temps d'exécution de l'algorithme dépend principalement de deux facteurs : le niveau maximal `niveau_max` et le nombre de sommets dans le groupe G_i calculé à chaque itération. Le premier paramètre correspond au nombre de groupes ou nombre de nœuds dans le premier O-Graphe og_1.

Ainsi, pour réduire la complexité, l'algorithme applique quelques techniques suivantes qui essaient de réduire les facteurs ci-dessus :

- *L'ordre des ontologies d'entrée* : [Durand *et al.,* 1999] montre l'ordre des structures (graphes) d'entrée utilisées pour créer le graphe d'association influence le temps d'exécution de l'algorithme de recherche de la clique maximale. La raison est que la taille du sous-graphe commun maximal trouvé ne dépasse jamais la taille (le nombre de nœuds) du graphe le plus petit de deux graphes d'entrée. Alors, si l'on prend le plus petit graphe (à propos du nombre de nœuds dans le graphe) comme le premier graphe, le nombre de groupes G_i (groupes de sommets du graphe d'association) est au maximum égal au nombre de nœuds du petit graphe. Ainsi, les ontologies à aligner sont remises en bon ordre pour que le plus petit O-Graphe correspondant à la première ontologie devienne le premier paramètre de l'*Algorithme 21* (`Construction_Graphe_Association`).

- *L'ordre des sommets dans un groupe* G_i : plus l'on trouve la taille de la clique maximale tôt, plus des branches dans l'arbre de recherche qui mènent des résultats ayant des cliques plus petites que la clique maximale jusqu'alors sont coupées tôt. Autrement dit, il faut que la valeur de `meilleur_nombre_sommets_nuls` s'approche la valeur réelle le plus tôt possible. En général, il y a plus de possibilité d'être un sommet dans la clique maximale pour les sommets ayant le nombre de connexions plus élevé que ceux qui ont moins de connexions dans le graphe d'association. Alors, la clique maximale est trouvée plus tôt si l'on considère dès le début les sommets du graphe d'association ayant le nombre de connexions le plus élevé. Ainsi, les sommets dans les groupes G_i sont triés en ordre descendant de leur nombre de connexions.

- *L'ordre des groupes* G_i : de même raison ci-dessus, les groupes G_i sont triés en ordre descendant de nombre de connexions du premier sommet dans chaque groupe.

- *Nombre de sommets dans chaque groupe* G_i : on peut utiliser des mesures de similarité utilisées dans les algorithmes ASCO1 ou ASCO2 pour éliminer des sommets dans les groupes G_i qui n'appartiennent sûrement pas à la clique

maximale. Notons que chaque sommet dans chaque groupe est une correspondance possible entre deux ontologies à aligner. Alors, si deux ontologies sont assez proches linguistiquement, et si parmi d'autres sommets dans un groupe G_i, on trouve des sommets se composant deux entités de deux ontologies qui sont linguistiquement similaires (leur valeur de similarité linguistique, calculée par des mesures linguistiques de ASCO1 ou ASCO2, dépasse un seuil T_L prédéfini), on peut ne garder que ces sommets dans le groupe G_i (les autres sommets sont enlevés du groupe). Cela diminue le nombre de correspondances possibles à traiter. Cependant, pas comme les autres techniques ci-dessus, cette technique peut renvoyer le résultat incorrect (la clique maximale trouvée n'est pas la clique maximale en réel) en enlevant de bonnes correspondances (des sommets corrects qui devraient être dans la clique maximale) à cause des différences dans la modélisation des ontologies au niveau linguistique et en choisissant une mauvaise valeur pour le seuil T_L.

- *De bonnes correspondances* : on peut utiliser le résultat renvoyé par les algorithmes ASCO1 ou ASCO2 pour réduire le nombre de groupes G_i ainsi que le nombre de sommets dans chaque groupe. Toutes les correspondances dans la liste de correspondances renvoyée par ASCO1 ou ASCO2, en leur fournissant un seuil de similarité prédéfini, sont considérées comme des correspondances correctes. Alors, les sommets du graphe d'association correspondent à ces correspondances appartiendront à la clique maximale. La recherche de la clique maximale commencera à partir de cet ensemble. Le nombre de groupes G_i diminue car l'on a enlevé tous les groupes contenant un sommet dans l'ensemble. Les bonnes correspondances entre deux ontologies à aligner peuvent être fournies également par les utilisateurs. Comme la dernière technique ci-dessus, cette technique peut renvoyer le résultat incorrect si les correspondances obtenues des algorithmes ASCO1 ou ASCO2, ou fournies par les utilisateurs, ne sont pas des correspondances correctes.

3.3.2 Conclusion

Dans cette section, nous avons présenté l'algorithme ASCO3 permettant de chercher des entités correspondantes entre deux ontologies en OWL, en exploitant le langage d'ontologie OWL. L'idée principale de ASCO3 est de représenter les ontologies en OWL à comparer par un graphe du type O-Graphe, c'est-à-dire un *réseau sémantique*. Ce réseau reflète la conceptualisation et la modélisation de l'ontologie, la façon dont les entités sont connectées, les liens sémantiques (correspondant à des primitives de OWL) entre des entités. Plus les deux réseaux sémantiques représentant deux ontologies sont proches, plus les deux ontologies en question sont similaires. L'algorithme ASCO3 cherche donc le sous-graphe commun le plus grand des deux graphes O-Graphe représentant les ontologies. Le problème se traduit dans un problème de recherche de la clique maximale du graphe d'association construit à partir des deux graphes O-Graphe.

La construction du graphe d'association prend en compte la sémantique des primitives de OWL via la notion de « *saut* ». Les nœuds et les arcs du graphe d'association sont créés non seulement par des relations directes entre les nœuds des O-Graphes, mais

aussi en exploitant les nœuds reliés indirectement par des primitives de `OWL` telles que `owl:subClassOf`, `owl:equivalentProperty`…

Comme prévu, les expérimentations dans le chapitre 4 montrent que l'algorithme ASCO3 retourne le meilleur résultat dans le cas où les ontologies d'entrée sont modélisées de manière expressive en employant plusieurs primitives de `OWL`. Le résultat ne sera pas très intéressant si les ontologies sont moins expressives, modélisées avec peu de primitives ou seulement quelques primitives, telles que `rdfs:subClassOf`, `rdfs:subPropertyOf`…

Les relations entre l'algorithme ASCO3 et les autres approches d'alignement des ontologies avec les mesures de similarité utilisées (qui sont résumées dans la *Figure 5*, page 19) sont montrées dans la *Figure 24*. Nous voyons que ASCO3 exploite certaines informations structurelles (voisines, isomorphisme du graphe) et certaines techniques et mesures comme ASCO1, ASCO2 (traitement automatique de la langue naturelle, Jaro-Winkler, TF/IDF) pour déduire des correspondances entre deux ontologies.

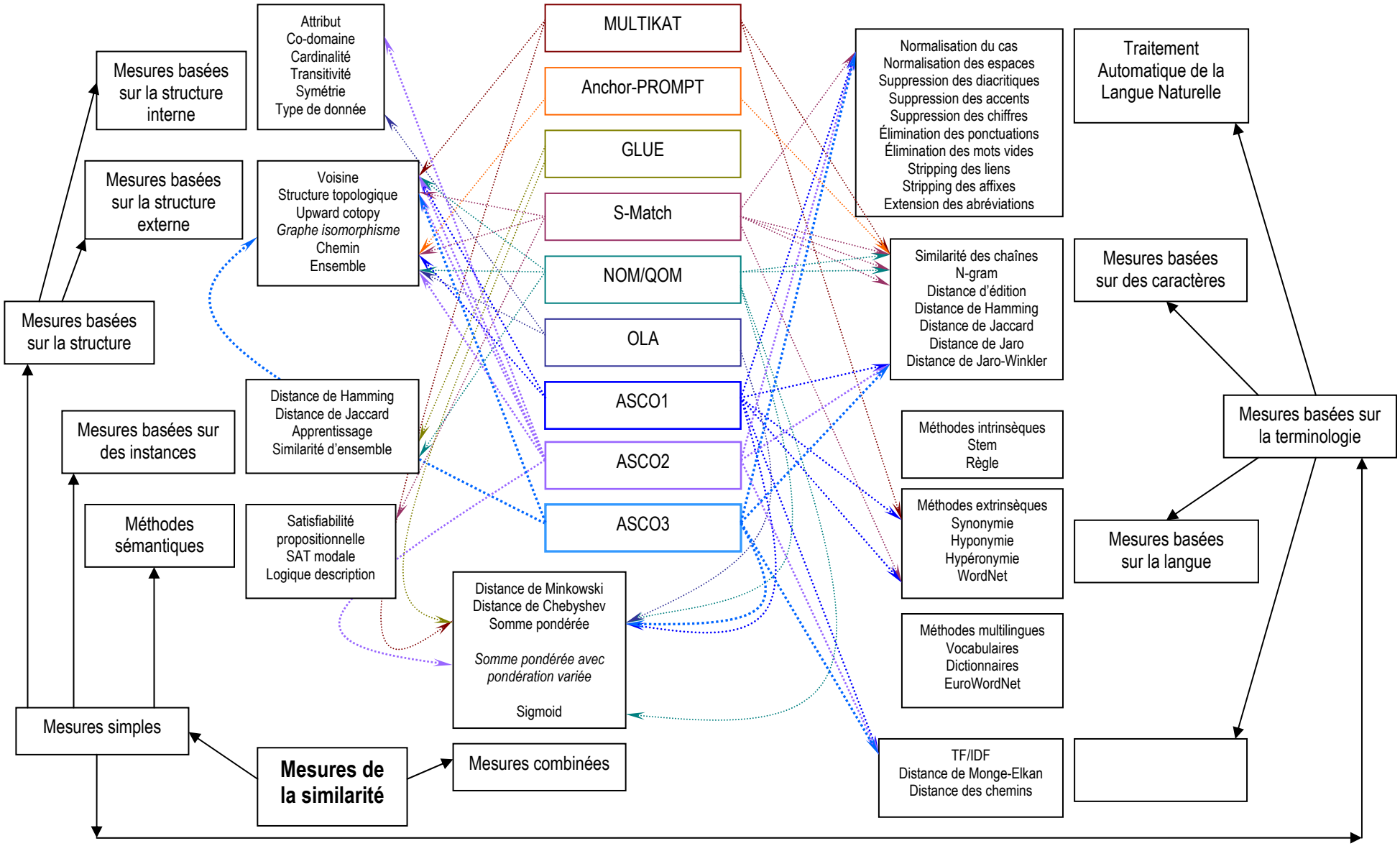


Figure 24 Le rapport entre l'algorithme ASCO3 et les autres approches d'alignement des ontologies

3.4 Conclusions

La tâche de mettre en correspondance des ontologies, ou l'alignement d'ontologies, dans ces dernières années a été devenu une tâche très émergente. Le résultat de cette tâche assure et facilite l'échange, le partage, la diffusion des données et des informations entre des systèmes ou des communautés dans le Web sémantique. Nous avons proposé trois algorithmes qui attaquent cette importante tâche dans le chapitre 2 et le chapitre 3.

L'algorithme ASCO1, présenté dans le chapitre 2, prend comme entrée deux ontologies en `RDF(S)` et renvoie une liste des entités correspondantes entre ces deux ontologies. Les correspondances renvoyées sont calculées en se basant sur les similarités des entités, qui sont déduites à partir des caractéristiques de langage `RDF(S)`. Les similarités sont calculées en employant des techniques et mesures telles que des mesures de similarité des chaînes de caractères via la distance linguistique, la technique TF/IDF, les heuristiques de similarité des structures.

Le chapitre 3 présente deux algorithmes traitant les ontologies en `OWL DL/Lite`, nommés ASCO2 et ASCO3. Ces algorithmes essaient d'exploiter les avantages d'expressivité du langage d'ontologie `OWL` pour déduire la similarité de deux entités dans les ontologies en `OWL`. ASCO2 prend en compte principalement les informations dans la *structure locale* des entités des ontologies, contenues dans leurs descriptions, pour déduire la similarité entre les entités. Il combine des similarités partielles obtenues de la comparaison des composantes dans des définitions des entités en employant le *schéma de pondération variable*. Quant à ASCO3, il analyse les rapports sémantiques entre des entités dans des ontologies (*structure globale*) pour extraire des correspondances entre des ontologies. ASCO3 représente les ontologies en `OWL` par des *réseaux sémantiques* et puis cherche le *sous-graphe commun le plus large* de ces réseaux en prenant en compte la sémantique des arcs (en introduisant la notion de « *saut* »). Les correspondances sont déduites du sous-graphe trouvé.

Dans le chapitre suivant, nous allons présenter les implémentations et les évaluations des algorithmes d'alignement d'ontologies proposés (ASCO1, ASCO2, ASCO3).

Chapitre 4

Implémentation et évaluation

Dans ce chapitre, nous présentons les implémentations et les évaluations des algorithmes d'alignement d'ontologies que nous avons proposés et présentés dans les chapitres 2 et 3. L'implémentation des algorithmes a été effectuée en Java et les évaluations ont été faites (1) sur notre base de tests, qui se compose de deux ontologies en RDF(S) nommées O'CoMMA et O'Aprobatiom; et (2) sur des bases de tests standard publiées à http://www.atl.external.lmco.com/projects/ontology/ et http://oaei.ontology matching. org/

4.1 Évaluation des algorithmes d'alignement d'ontologie

Les algorithmes d'alignement d'ontologies ASCO1, ASCO2, ASCO3 sont implémentés en Java, dans l'application nommée ASCO, pour évaluer leurs performances. Les fichiers des ontologies en `RDF(S)` ou `OWL` sous format de XML ou N3 sont parsés et chargés dans l'application ASCO en employant Jena[1] (celui qui fournit un environnement programmatique pour les applications du Web sémantique). Deux ontologies sont chargées à la fois et sont affichées dans deux fenêtres, à gauche et à droite (*Figure 25*). La structure de l'ontologie est affichée sous forme d'un arbre, avec la possibilité d'avoir plusieurs représentations d'une entité à plusieurs branches de l'arbre. Cela permet de représenter le multi-héritage d'une entité. Les liens entre des nœuds d'une branche sont les liens de spécialisation ou généralisation entre des entités. Il y a trois grands sous-arbres représentant une ontologie : un sous-arbre avec la racine nommée CLASSE_ROOT pour repésenter les classes, un sous-arbre avec la racine nommée PROPERTY_ROOT pour les propriétés et le troisième avec la racine nommée INSTANCE_ROOT pour les instances. Notons qu'il n'y a pas de lien de spécialisation entre les instances, le troisième sous-arbre a donc la profondeur maximale de 1. Toutes les instances dans l'ontologie sont les enfants directs de la racine INSTANCE_ROOT.

4.1.1 L'interface de l'application ASCO

La *Figure 25* est l'interface de l'application ASCO en cours (l'algorithme ASCO2 est activé) avec deux ontologies dans le domaine de l'hôtellerie chargées. Ce sont deux ontologies en `OWL` de la campagne de tests I3CON[2]. La fenêtre à gauche affiche l'ontologie hotelA.owl, avec 10 classes, 3 propriétés et 7 instances. La fenêtre à droite affiche l'ontologie hotelB.owl, avec 8 classes, 6 propriétés et 10 instances. Les informations à propos d'une entité sont affichées dans les fenêtres « Entity Infos » : son URI, son nom local, son type, et ses autres descriptions. La fenêtre « Mappings » affiche les correspondances trouvées par l'algorithme et celles qui sont dites correctes par les experts de domaine. Les chiffres entre 0 et 1 dans les correspondances sont les valeurs de similarité totale entre deux entités, calculées selon l'algorithme. La fenêtre « Benchmark » affiche des informations d'évaluation de l'algorithme par rapport aux correspondances correctes fournies par les experts : nombre des correspondances trouvées par l'algorithme (F – *Found*), nombre des correspondances correctes trouvées (C – *Correct*), nombre des correspondances incorrectes trouvées (I – *Incorrect*), et nombre des correspondances correctes (T – *True*). Enfin, la fenêtre « AlgoInfo » affiche des informations concernant les mesures de similarité de l'algorithme.

[1] http://jena.sourceforge.net/

[2] http://www.atl.lmco.com/projects/ontology/i3con.html

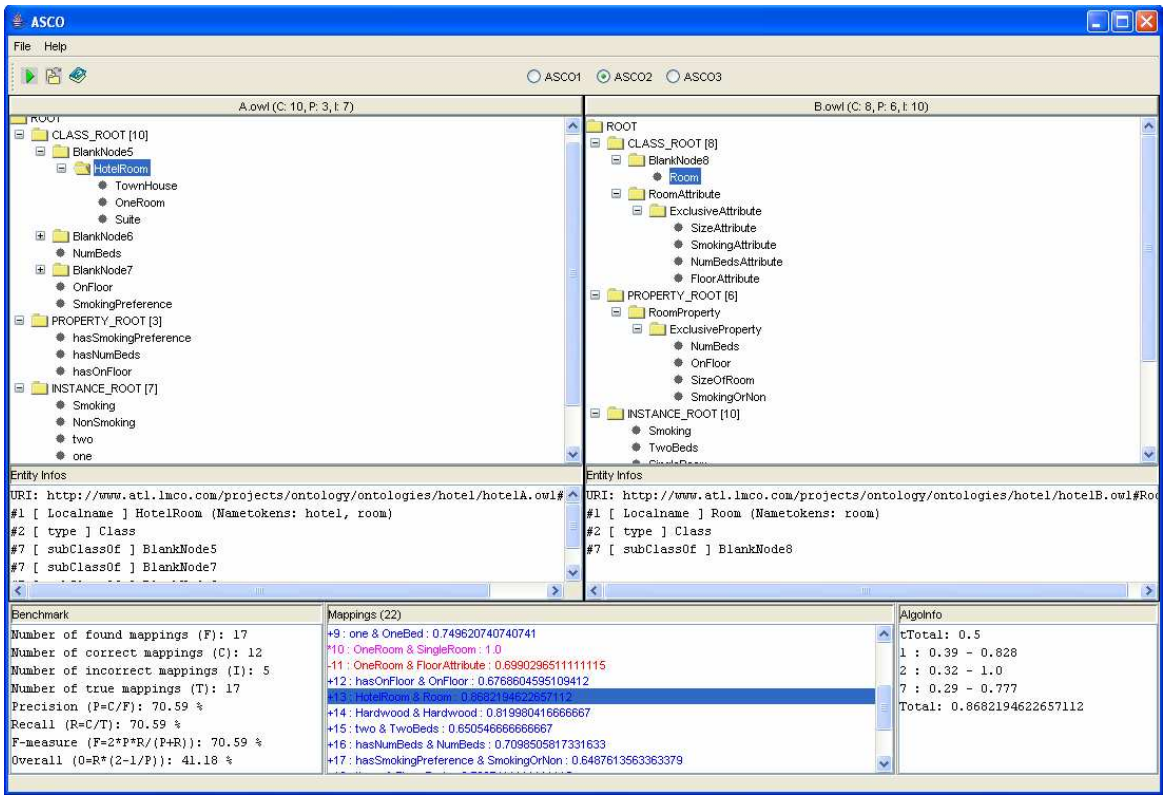


Figure 25 L'interface de l'application ASCO

Nous pouvons choisir l'algorithme par les boutons radio et le lancer grâce au bouton « play » .

L'application ASCO emploie un fichier nommé asco.config pour stocker des paramètres des algorithmes, tels que : le chemin où les fichiers d'ontologie se trouvent, les noms des ontologies d'entrée, les paramètres des poids pondérés pour les algorithmes, les seuils de similarité… En lançant, ASCO lit ce fichier de configuration, charge les ontologies indiquées, initialise les algorithmes et assigne les paramètres dans le fichier auprès les paramètres des algorithmes. Le bouton permet de relire le fichier de configuration. Il est donc possible de modifier des paramètres dans le fichier (tels que changer les noms d'ontologie d'entrée, varier des poids pondérés) et recharger ces paramètres modifiés quand l'application ASCO est en cours.

L'application ASCO charge aussi un autre fichier de référence (benchmark) qui contient des correspondances entre deux ontologies. Les correspondances sont détectées et confirmées comme correctes par des experts du domaine. Elles sont stockées internement dans l'application sous forme des paires de URI d'entité. ASCO peut lire des fichiers de référence en trois différents formats :

Le **format d'ASCO** : Les correspondances sont stockées en chaque ligne dans le fichier de référence, sous forme :

```
URI_entité1 <==> URI_entité2
```

où URI_entité1 et URI_entité2 sont les URIs de l'entité dans la première et la deuxième ontologie, respectivement. Chaque ligne du fichier de référence contient une paire d'entités, qui sont considérées similaires selon des experts.

Le **format d'I3CON** et le **format d'OAEI** : ce sont des formats utilisés dans les campagnes de test : I^{3}CON (Information Interpretation and Integration Conference[1]) et OAEI (Ontology Alignment Evaluation Initiative[2]). Ces campagnes fournissent des paires d'ontologies à aligner, accompagnant avec les correspondances correctes entre deux ontologies dans les fichiers de référence dans leurs propres formats. L'application ASCO permet de faire aligner ces paires d'ontologies dans ces campagnes et utilise leurs fichiers de référence fournis pour évaluer la performance des algorithmes.

4.1.2 Mesures d'évaluation

Le résultat des algorithmes ASCOx est une liste des triplets <`entité1` `entité2` `sim`> où `entité1` et `entité2` sont les URIs des entités dans deux ontologies, qui sont trouvées par l'algorithme comme similaires, et `sim` est la valeur de similarité entre ces deux entités, calculée par des mesures de l'algorithme. Les correspondances trouvées sont comparées avec les correspondances correctes spécifiées dans le fichier de référence. Si une paire d'entités trouvée par l'algorithme est correcte (elle existe dans le fichier de référence), la correspondance est affichée en bleu avec le symbole « + » dans la fenêtre « Mapping ». Si une paire d'entités trouvée n'est pas correcte, la correspondance est représentée en rouge avec le symbole « - ». Pour toutes les correspondances correctes que l'algorithme n'a pas détectées, elles sont représentées en magenta avec le symbole « * » (*Figure 25*).

Pour évaluer la précision, l'efficacité et la performance de l'algorithme, nous employons les mesures de précision, de rappel et de f-mesure. Ce sont les mesures bien utilisées dans le domaine de recherche d'information et ensuite appliquées dans le domaine d'alignement d'ontologies pour permettre une analyse fine des performances de système.

Le *rappel* est la proportion de correspondances correctes renvoyées par l'algorithme parmi toutes celles qui sont correctes (en incluant aussi des correspondances correctes que l'algorithme n'a pas détectées). Le rappel mesure l'efficacité d'un algorithme. Plus la valeur de rappel est élevée, plus le résultat de l'algorithme couvre toutes les correspondances correctes.

La *précision* est la proportion des correspondances correctes parmi l'ensemble de celles renvoyées par l'algorithme (ce sont les correspondances dans la liste des quadruplet). Cette mesure reflète la précision d'un algorithme. Plus la valeur de précision est élevée, plus le bruit dans le résultat de l'algorithme est réduit, et donc plus la qualité de résultat est imposante.

---

[1] http://www.atl.lmco.com/projects/ontology/i3con.html

[2] http://oaei.ontologymatching.org

Enfin, la *f-mesure* est un compromis entre le rappel et la précision. Elle permet de comparer les performances des algorithmes par une seule mesure. La f-mesure est définie par

$$f - mesure = \frac{2 * rappel * precision}{rappel + precision}$$

Nous utilisons aussi la *mesure globale* (overall measure) définie dans [Melnik *et al.*, 2002]. Cette mesure, appelée l'exactitude, correspond à l'effort exigé pour corriger le résultat renvoyé par l'algorithme afin d'obtenir le résultat correct. La mesure globale est toujours inférieure à la f-mesure. Elle n'a un sens que dans le cas où la précision n'est pas inférieure à 0,5, c'est-à-dire si au moins la moitié des correspondances renvoyées par l'algorithme sont correctes. En effet, si plus d'une moitié des correspondances sont erronées, il faudrait à l'utilisateur plus d'effort d'enlever les correspondances incorrectes et d'ajouter les correspondances correctes mais absentes, que pour mettre en correspondance manuellement les deux ontologies à partir de zéro.

$$overall = rappel * \left(2 - \frac{1}{precision}\right)$$

Nous notons :

- F – le nombre des correspondances renvoyées par l'algorithme
- T – le nombre des correspondances correctes déterminées manuellement par des experts (celles dans le fichier de référence)
- C – le nombre des correspondances correctes trouvées (appelé aussi vrais positifs)
- I = F - C – nombre des correspondances incorrectes trouvées (appelé aussi faux positifs)
- M = T - C – nombre des correspondances correctes mais pas trouvées (appelé aussi faux négatifs).

Ainsi, *précision* = C / F ; *rappel* = C / T ; *f-mesure* = 2*P*R/(P+R) et *overall* = R*(2-1/P).

4.1.3 Méthodologie d'évaluation

Nous avons testé les algorithmes proposés avec les paires d'ontologies de test dans deux campagnes : I^{3}CON et OAEI. Pour chaque paire d'ontologie, ASCO1 et ASCO2 sont exécutés 6 fois avec 6 seuils différents de similarité. Les autres paramètres des algorithmes (tels que les poids pondérés) sont inchangés. Le seuil est utilisé dans les algorithmes pour déterminer des correspondances considérées comme correctes. La valeur de similarité totale de deux entités de deux ontologies, calculée par des mesures linguistiques et structurelles, est comparée avec le seuil. Deux entités sont considérées comme similaires si leur valeur de similarité dépasse ce seuil. Le résultat final de l'algorithme (la liste des correspondances entre deux ontologies) dépend donc du seuil choisi. Plus le seuil de similarité est élevé, plus la qualité du résultat est élevée (la précision augmente, le nombre des correspondances incorrectes trouvées diminue) mais plus des correspondances correctes sont considérées comme incorrectes. En continuant à

augmenter le seuil de similarité, les valeurs de f-mesure et d'overall diminuent car le nombre des correspondances correctes mais pas trouvées devient très important. La raison est qu'il y a des entités qui représentent des concepts similaires mais leurs définitions dans des ontologies sont un peu différentes, donc leur valeur de similarité, calculée en se basant sur leurs définitions, n'atteint pas une valeur élevé (le maximum est 1.0), et alors ne dépasse pas le seuil.

Les correspondances trouvées par l'algorithme sont comparées avec les correspondances correctes dans le fichier de référence (fourni aussi par les campagnes de test), pour calculer le nombre des correspondances correctes et incorrectes trouvées par l'algorithme (T et I, respectivement). Les valeurs des mesures d'évaluation (précision, rappel, f-mesure, overall) sont calculées pour chaque cas et affichées dans un tableau et un graphique. L'algorithme ASCO3 cherche des correspondances entre deux ontologies par la détection de la clique maximale dans le graphe d'association, il n'utilise donc pas des seuils de similarité pour décider les entités similaires. Par contre, nous varions les paramètres `nRadius_Super` et `nRadius_Sub` pour voir leur influence sur le résultat renvoyé par l'algorithme.

Nous avons testé les algorithmes avec les ontologies dans notre propre plate-forme d'essai, qui sont une paire d'ontologies réelles en `RDF(S)` O'COMMA et O'Aprobatiom ; et d'autres paires de tests pour analyser des comportements des algorithmes.

4.1.3.1 Évaluation des paires d'ontologies dans la campagne I^{3}CON

I^{3}CON (Information Interpretation and Integration Conference)[1] est le premier effort dans la communauté de la recherche de l'intégration et de l'interprétation de l'ontologie et du schéma, pour fournir des outils et un framework d'évaluation systématique de l'intégration des ontologies et des schémas. I^{3}CON fournit 10 paires d'ontologies accompagnées de leur fichier de référence qui contient les correspondances déterminées manuellement entre deux ontologies. Le *Tableau 10* résume des informations concernant les paires d'ontologies de test, avec leur nom, leur langage de représentation, leur format de sérialisation en fichier, le nombre de classes, de propriétés, d'instances dans les deux ontologies, et le nombre de correspondances correctes dans le fichier de référence fourni.

[1] http://www.atl.lmco.com/projects/ontology/i3con.html

Paire d'ontologie	Langage	Format	Nombre de classes	Nombre de propriétés	Nombre d'instances	Nombre de corr. correctes
Animals	OWL	RDF/XML	13-13	15-14	11-0	24
Sports	DAML	RDF/XML	-	-	-	112
Computer Science	DAML	RDF/XML	-	-	-	16
Hotels	OWL	RDF/XML	10-8	3-6	7-10	17
Computer Networks	OWL	RDF/XML	27-27	5-6	0-2	30
Pets	OWL	RDF/XML	121-113	15-15	21-20	93
Pets (no instances)	OWL	RDF/XML	120-112	15-15	0-0	74
Russia	RDF	RDF/XML	162-151	80-75	214-158	117
Wine	OWL	N3	33-33	0-0	0-0	33
Weapons	OWL	N3	79-81	0-0	0-0	73

Tableau 10 Les paires d'ontologies dans la campagne de tests I³CON

Parmi ces paires d'ontologies, la paire d'ontologies « *Sports* » et « *Computer Science* » sont en langage DAML. Elles ne sont donc pas évaluées par les algorithmes ASCO car ces derniers sont conçus pour faire aligner les ontologies en `RDF` et en `OWL`. Les primitives et les classes du langage DAML telles que `daml:label`, `daml:Class`… ne sont pas reconnues par ASCO.

4.1.3.1.1 La paire d'ontologies « Animals »

Les deux ontologies de cette paire sont très proches au niveau structurel et aussi bien au niveau linguistique. La première a 11 instances tandis que la deuxième n'a aucune instance, mais cela n'influence pas la performance des trois algorithmes ASCO. La différence entre ces deux ontologies est la définition des classes et des propriétés équivalentes. Par exemple, la première ontologie définit `hasFemaleParent` comme la propriété équivalente de la propriété `hasMother`, et `HumanBeing` comme la classe équivalente de la classe `Person`. L'application ASCO exécute un post-traitement permettant de détecter des entités équivalentes. Cela permet de trouver toutes les correspondances de ce type. Les résultats renvoyés par les algorithmes sont très bons. Avec les seuils de similarité moins de 0,82 pour ASCO1 et 0,5 pour ASCO2, toutes les correspondances correctes sont bien trouvées. Si l'on augmente ces seuils, le nombre des correspondances correctes mais non-détectées est aussi augmenté, et donc, la valeur de f-mesure diminue.

Seuil	F	T	C	I=F-C	M=T-C	Précision P=C/F	Rappel R=C/T	F-mesure FM=2*P*R/(P+R)	Overall O=R*(2-1/P)
0,70	24	24	24	0	0	1.000	1.000	1.000	1.000
0,80	24	24	24	0	0	1.000	1.000	1.000	1.000
0,82	24	24	24	0	0	1.000	1.000	1.000	1.000
0,85	19	24	19	0	5	1.000	0.792	0.884	0.792
0,90	18	24	18	0	6	1.000	0.750	0.857	0.750
1,00	15	24	15	0	9	1.000	0.625	0.769	0.625

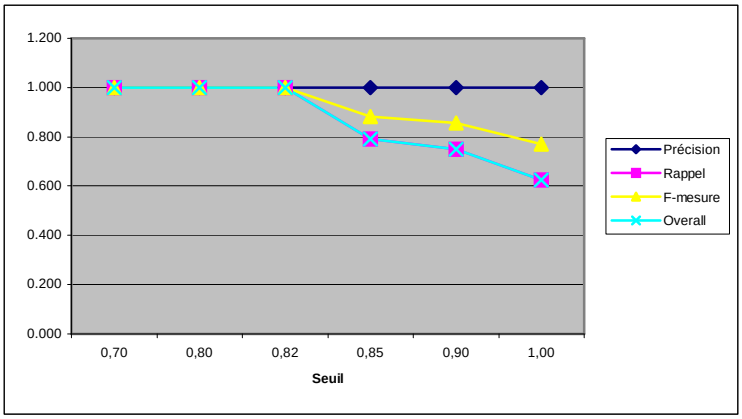


Figure 26 Résultat de ASCO1 sur la paire d'ontologies « Animals »

Seuil	F	T	C	I=F-C	M=T-C	Précision P=C/F	Rappel R=C/T	F-mesure FM=2*P*R/(P+R)	Overall O=R*(2-1/P)
0,50	24	24	24	0	0	1.000	1.000	1.000	1.000
0,60	21	24	21	0	3	1.000	0.875	0.933	0.875
0,70	21	24	21	0	3	1.000	0.875	0.933	0.875
0,80	18	24	18	0	6	1.000	0.750	0.857	0.750
0,90	11	24	11	0	13	1.000	0.458	0.629	0.458
0,95	6	24	6	0	18	1.000	0.250	0.400	0.250

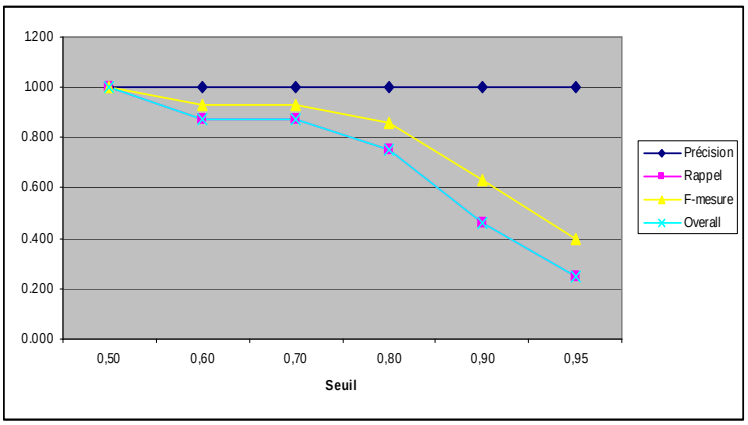


Figure 27 Résultat de ASCO2 sur la paire d'ontologies « Animals »

L'algorithme ASCO3 renvoie aussi un résultat parfait dans ce test.

nRadius_Super nRadius_Sub	F	T	C	I=F-C	M=T-C	Précision P=C/F	Rappel R=C/T	F-mesure FM=2*P*R/(P+R)	Overall O=R*(2-1/P)
1-1	24	24	24	0	0	1.000	1.000	1.000	1.000
1-2	24	24	24	0	0	1.000	1.000	1.000	1.000
2-1	24	24	24	0	0	1.000	1.000	1.000	1.000
2-2	24	24	24	0	0	1.000	1.000	1.000	1.000

Tableau 11 Résultat de ASCO3 sur la paire d'ontologies « Animals »

4.1.3.1.2 La paire d'ontologies « Hotels »

Les deux ontologies de cette paire ont une structure assez différente et les entités sont nommées différemment. Par exemple, la classe `OnFloor` de la première ontologie correspond à la classe `FloorAttribute` de la deuxième ontologie ; la propriété `hasSmokingPreference` de la première ontologie qui n'a aucune propriété parent et donc seul le domaine est défini (son co-domaine n'est pas défini) est déterminée similaire à la propriété `SmokingOrNon` de la deuxième ontologie, qui est sous-propriété de la propriété `ExclusiveProperty`, et est définie avec son domaine (la classe `Room`) et son co-domaine (la classe `SmokingAttribute`). Cependant, ASCO1 et ASCO2 peuvent toujours détecter ces correspondances si le seuil de similarité est bien choisi. Par exemple, la valeur de similarité totale entre `OnFloor` et `FloorAttribute` calculée par ASCO1 est de 0,786. Pour tous les seuils inférieurs à cette valeur, cette correspondance est toujours trouvée. La valeur de f-mesure est maximale à 0,813 et 0,706 pour le seuil de similarité de 0,75 et 0,6 dans ASCO1 et ASCO2, respectivement. Le résultat dans ce test n'est pas très bon en raison des correspondances correctes définie dans le fichier de référence. Trois paires d'entités, entre une classe et une instance, sont déterminées comme les correspondances correctes. Par exemple, la *classe* `OneRoom` de la première ontologie est déterminée correspondante avec l'*instance* `SingleRoom` de la deuxième ontologie. Les algorithmes d'ASCO ne mettent en correspondance que des entités de même nature entre deux ontologies. Ils ne peuvent pas donc détecter des correspondances de ce type.

Seuil	F	T	C	I=F-C	M=T-C	Précision P=C/F	Rappel R=C/T	F-mesure FM=2*P*R/(P+R)	Overall O=R*(2-1/P)
0,50	17	17	13	4	4	0.765	0.765	0.765	0.529
0,60	17	17	13	4	4	0.765	0.765	0.765	0.529
0,70	17	17	13	4	4	0.765	0.765	0.765	0.529
0,75	15	17	13	2	4	0.867	0.765	0.813	0.647
0,80	12	17	10	2	7	0.833	0.588	0.690	0.471
0,85	7	17	7	0	10	1.000	0.412	0.583	0.412

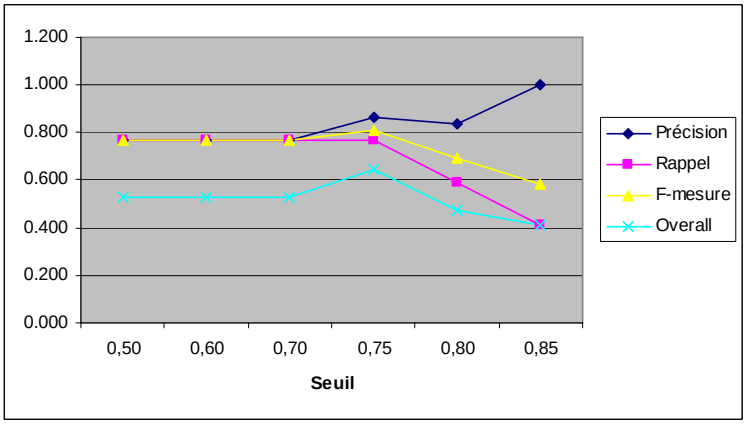


Figure 28 Résultat de ASCO1 sur la paire d'ontologies « Hotels »

Seuil	F	T	C	I=F-C	M=T-C	Précision P=C/F	Rappel R=C/T	F-mesure FM=2*P*R/(P+R)	Overall O=R*(2-1/P)
0,50	17	17	12	5	5	0.706	0.706	0.706	0.412
0,55	17	17	12	5	5	0.706	0.706	0.706	0.412
0,60	17	17	12	5	5	0.706	0.706	0.706	0.412
0,65	15	17	10	5	7	0.667	0.588	0.625	0.294
0,70	11	17	8	3	9	0.727	0.471	0.571	0.294
0,80	5	17	5	0	12	1.000	0.294	0.455	0.294

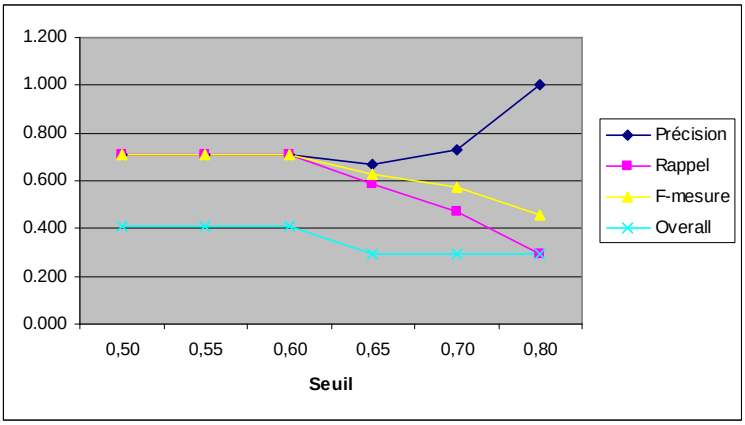


Figure 29 Résultat de ASCO2 sur la paire d'ontologies « Hotels »

nRadius_Super nRadius_Sub	F	T	C	I=F-C	M=T-C	Précision P=C/F	Rappel R=C/T	F-mesure FM=2*P*R/(P+R)	Overall O=R*(2-1/P)
0-0	15	17	13	2	4	0.867	0.765	0.813	0.647
1-1	16	17	13	3	4	0.813	0.765	0.788	0.588
1-2	16	17	13	3	4	0.813	0.765	0.788	0.588
2-1	16	17	13	3	4	0.813	0.765	0.788	0.588
2-2	16	17	13	3	4	0.813	0.765	0.788	0.588

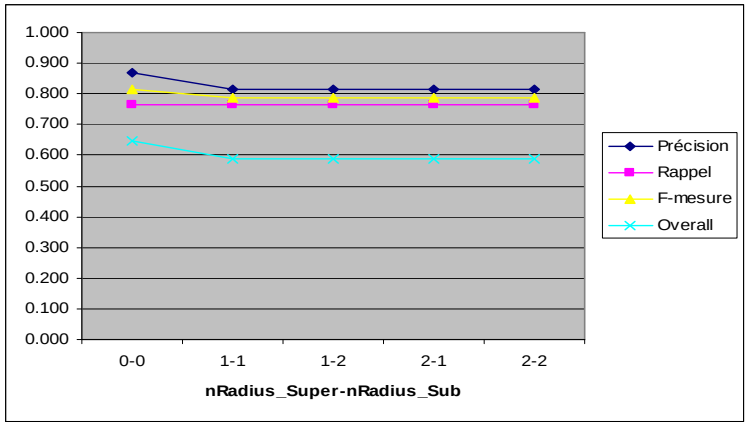


Figure 30 Résultat de ASCO3 sur la paire d'ontologies « Hotels »

4.1.3.1.3 La paire d'ontologies « Computer Networks »

Comme la paire d'ontologies « *Animals* », deux ontologies dans cette paire ont une structure similaire et les termes nommant les entités sont aussi similaires. Par contre, la structure d'ontologie est assez complexe, avec la profondeur maximale de 4. L'algorithme ASCO1 se base principalement sur la linguistique, il détecte incorrectement donc que la classe `FTPServer` de la première ontologie, qui est sous-classe de `ServerSoftware`, correspond à la classe `Server` de la deuxième ontologie, qui est sous-classe de `Computer`, car la valeur de similarité totale calculée est de 0,94. Tandis que la valeur de similarité entre `FTPServer` et sa vraie classe correspondante dans la deuxième ontologie, `FTP`, n'est que 0,916. L'algorithme ASCO2 prend en compte la structure d'ontologie et trouve donc correctement la classe correspondante de la classe `FTPServer`, qui est la classe `FTP` (sous-classe de `ServerSoftware`), car la valeur de similarité selon ASCO2, entre `FTPServer` et `FTP` est de 0,939 et entre `FTPServer` et `Server` est de 0,931.

La valeur de f-mesure est maximale à 0,769 et 0,929 avec le seuil de similarité de 0,925 et 0,9 dans ASCO1 et ASCO2, respectivement. Dans ce test, ASCO2 renvoie un résultat meilleur que ASCO1. La valeur de f-mesure pour ASCO3 est de 0.933 est la valeur meilleure. Le résultat obtenu n'est parfait dans aucun des trois algorithmes parce que dans le fichier de référence, nous avons 4 correspondances : `NodeA-NodeA`, `NodeA-NodeB`, `NodeB-NodeA`, `NodeB-NodeB`, tandis que les algorithmes proposés ne cherchent qu'une seule entité correspondante dans la deuxième ontologie pour chaque entité de la première ontologie. Ainsi, deux correspondances `NodeA-NodeB`, `NodeB-NodeA` ne sont pas détectées par les algorithmes.

Seuil	F	T	C	I=F-C	M=T-C	Précision P=C/F	Rappel R=C/T	F-mesure FM=2*P*R/(P+R)	Overall O=R*(2-1/P)
0,70	32	30	21	11	9	0.656	0.700	0.677	0.333
0,80	30	30	21	9	9	0.700	0.700	0.700	0.400
0,85	28	30	21	7	9	0.750	0.700	0.724	0.467
0,90	26	30	20	6	10	0.769	0.667	0.714	0.467
0,925	22	30	20	2	10	0.909	0.667	0.769	0.600
0,95	15	30	15	0	15	1.000	0.500	0.667	0.500

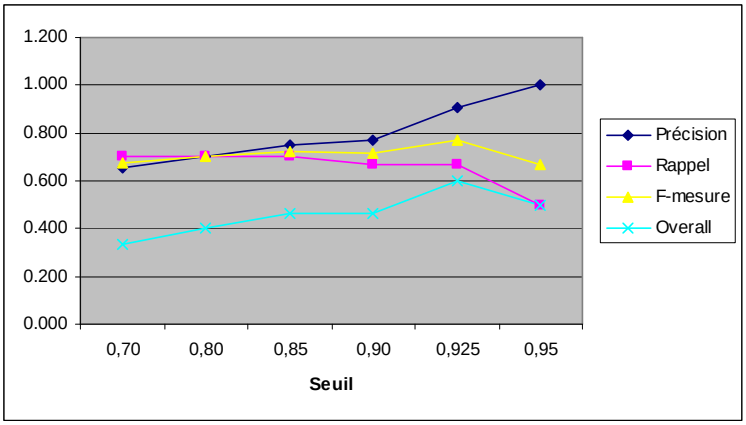


Figure 31 Résultat de ASCO1 sur la paire d'ontologies « Computer Networks »

Seuil	F	T	C	I=F-C	M=T-C	Précision P=C/F	Rappel R=C/T	F-mesure FM=2*P*R/(P+R)	Overall O=R*(2-1/P)
0,60	32	30	26	6	4	0.813	0.867	0.839	0.667
0,75	32	30	26	6	4	0.813	0.867	0.839	0.667
0,70	32	30	26	6	4	0.813	0.867	0.839	0.667
0,80	31	30	26	5	4	0.839	0.867	0.852	0.700
0,85	27	30	26	1	4	0.963	0.867	0.912	0.833
0,90	26	30	26	0	4	1.000	0.867	0.929	0.867

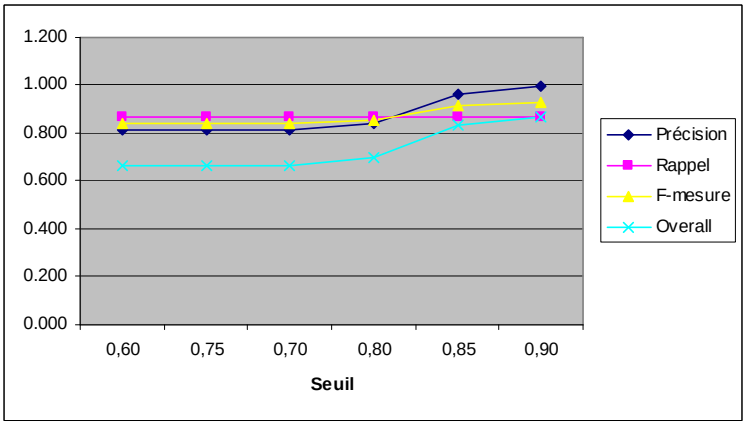


Figure 32 Résultat de ASCO2 sur la paire d'ontologies « Computer Networks »

nRadius_Super nRadius_Sub	F	T	C	I=F-C	M=T-C	Précision P=C/F	Rappel R=C/T	F-mesure FM=2*P*R/(P+R)	Overall O=R*(2-1/P)
0-0	30	30	28	2	2	0.933	0.933	0.933	0.867
1-1	30	30	28	2	2	0.933	0.933	0.933	0.867
2-2	30	30	28	2	2	0.933	0.933	0.933	0.867

Tableau 12 Résultat de ASCO3 sur la paire d'ontologies « Computer Networks »

4.1.3.1.4 La paire d'ontologies « Pets » et la paire d'ontologies « Pets (no instances) »

Deux paires d'ontologies contiennent des ontologies similaires. Pour la première paire, les ontologies ont une vingtaine instances. Pour la deuxième paire, les ontologies n'ont aucune instance. Les résultats obtenus dans ces cas de test pour les deux algorithmes ASCO1 et ASCO2 sont assez bons. Pour la paire d'ontologies « *Pets* », la valeur de f-mesure est maximale à 0,955 et 0,949 pour le seuil de similarité de 0,7 et 0,7 dans ASCO1 et ASCO2, respectivement. Pour la paire d'ontologies « *Pets (no instances)* », par coïncidence, la valeur de f-mesure est maximale à la même valeur de 0,935 pour le même seuil de similarité de 0,7 dans les deux algorithmes ASCO1 et ASCO2.

Certaines correspondances telles que `bus_company` et `coach_company` ne peuvent pas être détectées par la version d'implémentation actuelle des algorithmes car dans cette version, nous n'employons pas encore un dictionnaire de synonyme. Comme nous l'avons présenté dans la section *2.3.1.5*, des correspondances de ce type sont bien trouvées en intégrant des ressources externes de synonymes telles que WordNet ou EuroWordNet. Nous envisageons d'inclure un API (Application Programming Interface) en Java permettant accéder à la bibliothèque de WordNet[1], tels que JWordNet[2] ou JWNL[3], dans la version future de ASCO.

Pour ce test, nous n'avons pas eu le résultat de test pour l'algorithme ASCO3. Comme nous l'avons souligné dans la section *3.3.1.3*, la complexité de l'algorithme ASCO3 correspond à la complexité de l'algorithme de recherche de la clique maximale dans le graphe d'association, qui est O(`N`!) où `N` est nombre des nœuds du graphe d'association. Avec la version d'implémentation actuelle de l'algorithme ASCO3, qui n'est pas encore bien optimisée, nous ne pouvons tester l'algorithme qu'avec des ontologies ayant une trentaine d'entités au total. Si les ontologies se composent de beaucoup d'entités, le temps d'exécution de l'algorithme sera très important.

[1] http://wordnet.princeton.edu/links#Java

[2] http://www.seas.gwu.edu/~simhaweb/software/jwordnet/

[3] http://sourceforge.net/projects/jwordnet

Seuil	F	T	C	I=F-C	M=T-C	Précision P=C/F	Rappel R=C/T	F-mesure FM=2*P*R/(P+R)	Overall O=R*(2-1/P)
0,50	94	93	85	9	8	0.904	0.914	0.909	0.817
0,55	92	93	85	7	8	0.924	0.914	0.919	0.839
0,60	90	93	85	5	8	0.944	0.914	0.929	0.860
0,65	87	93	85	2	8	0.977	0.914	0.944	0.892
0,70	85	93	85	0	8	1.000	0.914	0.955	0.914
0,80	73	93	73	0	20	1.000	0.785	0.880	0.785

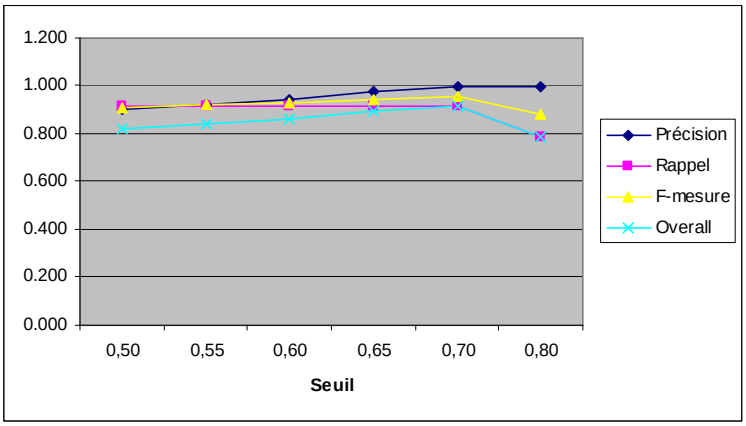


Figure 33 Résultat de ASCO1 sur la paire d'ontologies « Pets »

Seuil	F	T	C	I=F-C	M=T-C	Précision P=C/F	Rappel R=C/T	F-mesure FM=2*P*R/(P+R)	Overall O=R*(2-1/P)
0,50	95	93	86	9	7	0.905	0.925	0.915	0.828
0,55	95	93	86	9	7	0.905	0.925	0.915	0.828
0,60	90	93	85	5	8	0.944	0.914	0.929	0.860
0,65	87	93	84	3	9	0.966	0.903	0.933	0.871
0,70	84	93	84	0	9	1.000	0.903	0.949	0.903
0,80	51	93	51	0	42	1.000	0.548	0.708	0.548

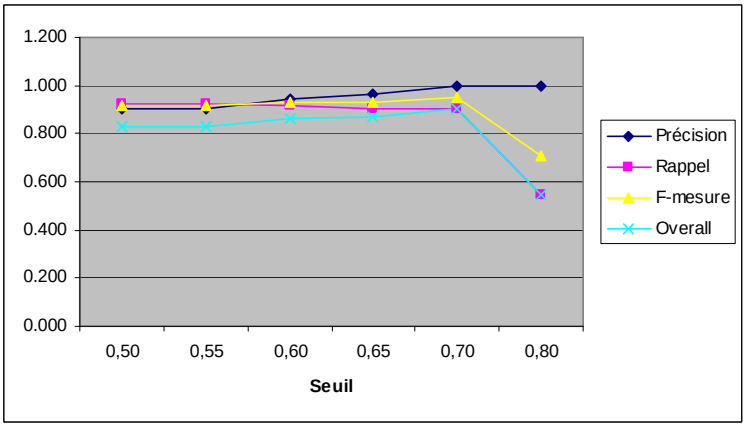


Figure 34 Résultat de ASCO2 sur la paire d'ontologies « Pets »

Seuil	F	T	C	I=F-C	M=T-C	Précision P=C/F	Rappel R=C/T	F-mesure FM=2*P*R/(P+R)	Overall O=R*(2-1/P)
0,50	73	74	65	8	9	0.890	0.878	0.884	0.770
0,55	71	74	65	6	9	0.915	0.878	0.897	0.797
0,60	69	74	65	4	9	0.942	0.878	0.909	0.824
0,65	66	74	65	1	9	0.985	0.878	0.929	0.865
0,70	65	74	65	0	9	1.000	0.878	0.935	0.878
0,80	53	74	53	0	21	1.000	0.716	0.835	0.716

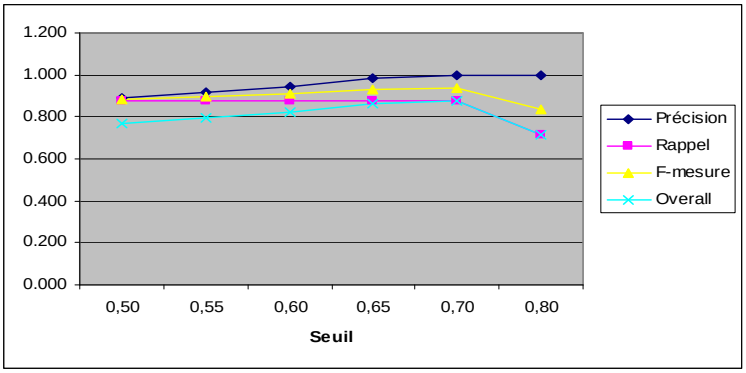


Figure 35 Résultat de ASCO1 sur la paire d'ontologies « Pets (no instances) »

Seuil	F	T	C	I=F-C	M=T-C	Précision P=C/F	Rappel R=C/T	F-mesure FM=2*P*R/(P+R)	Overall O=R*(2-1/P)
0,50	74	74	66	8	8	0.892	0.892	0.892	0.784
0,55	74	74	66	8	8	0.892	0.892	0.892	0.784
0,60	70	74	65	5	9	0.929	0.878	0.903	0.811
0,65	68	74	65	3	9	0.956	0.878	0.915	0.838
0,70	65	74	65	0	9	1.000	0.878	0.935	0.878
0,80	48	74	48	0	26	1.000	0.649	0.787	0.649

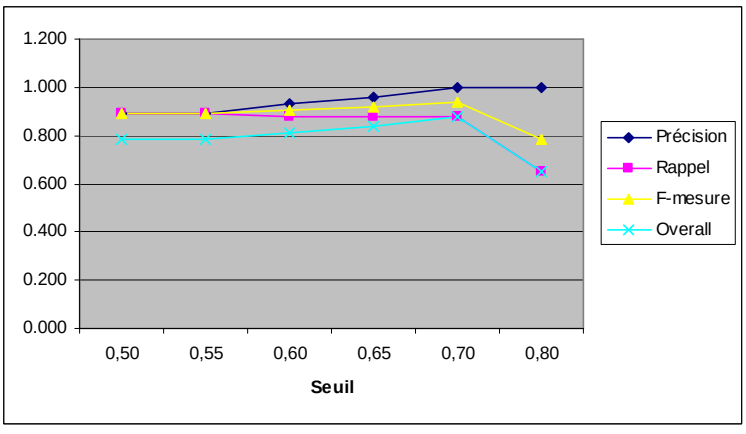


Figure 36 Résultat de ASCO2 sur la paire d'ontologies « Pets (no instances) »

4.1.3.1.5 La paire d'ontologies « Russia »

Les ontologies dans cette paire apparaissent comme les ontologies les plus difficiles à aligner dans cette campagne de test. Elles ont des structures très différentes. Aussi, les entités sont nommées de manière très différente. Ce sont des ontologies en `RDF` (et non en `OWL` comme les paires d'ontologies précédentes). Ce sont aussi de grandes ontologies : la première a 456 entités (dont 162 classes, 80 propriétés, 214 instances), et la deuxième a 384 entités (dont 154 classes, 75 propriétés, 158 instances). Nous pouvons trouver dans le fichier de référence fourni par I^3CON, que la propriété `has_sight_city` de la première ontologie correspond à la propriété `has_attractions` de la deuxième ontologie. En plus, la première propriété a pour domaine la classe `city` et pour co-domaine la classe `public_space`, tandis que pour la deuxième propriété, ce sont `country` et `city`, respectivement. Ce type de correspondance est très difficile à détecter automatiquement.

Cependant, les résultats renvoyés par les algorithmes ASCO1 et ASCO2 sont excellents. La valeur de f-mesure est maximale à 0,807 et 0,842 pour le seuil de similarité de 0,95 et 0,9 dans ASCO1 et ASCO2, respectivement.

Seuil	F	T	C	I=F-C	M=T-C	Précision P=C/F	Rappel R=C/T	F-mesure FM=2*P*R/(P+R)	Overall O=R*(2-1/P)
0,70	454	117	100	354	17	0.220	0.855	0.350	-2.171
0,75	408	117	100	308	17	0.245	0.855	0.381	-1.778
0,80	304	117	100	204	17	0.329	0.855	0.475	-0.889
0,85	194	117	99	95	18	0.510	0.846	0.637	0.034
0,90	130	117	95	35	22	0.731	0.812	0.769	0.513
0,95	101	117	88	13	29	0.871	0.752	0.807	0.641

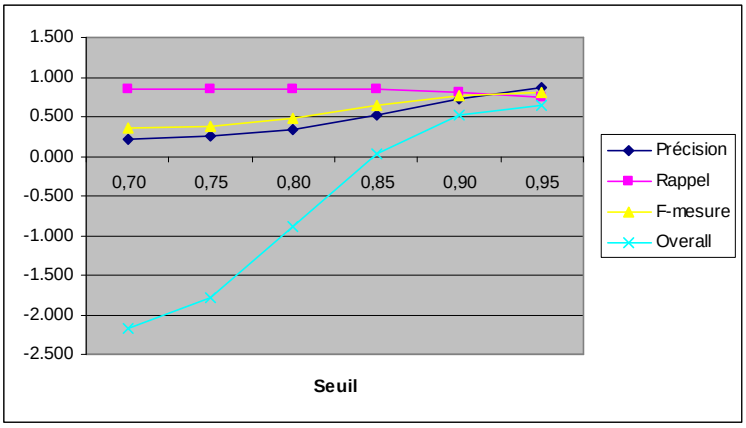


Figure 37 Résultat de ASCO1 sur la paire d'ontologies « Russia »

Seuil	F	T	C	I=F-C	M=T-C	Précision P=C/F	Rappel R=C/T	F-mesure FM=2*P*R/(P+R)	Overall O=R*(2-1/P)
0,70	363	117	98	265	19	0.270	0.838	0.408	-1.427
0,75	300	117	98	202	19	0.327	0.838	0.470	-0.889
0,80	195	117	97	98	20	0.497	0.829	0.622	-0.009
0,85	129	117	94	35	23	0.729	0.803	0.764	0.504
0,90	104	117	93	11	24	0.894	0.795	0.842	0.701
0,95	54	117	52	2	65	0.963	0.444	0.608	0.427

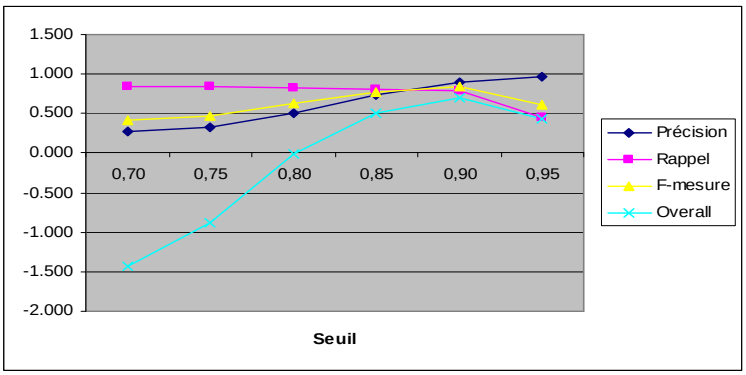


Figure 38 Résultat de ASCO2 sur la paire d'ontologies « Russia »

4.1.3.1.6 La paire d'ontologies « Wine »

Les ontologies dans cette paire sont représentées en `OWL` et sont stockées en format `N3`. Elles n'ont que des classes (33 au total), et pas de propriétés ni d'instances. On trouve des correspondances entre deux ontologies dont les noms des classes sont encodés en langue différente (anglais et italien) : par exemple, la classe `WhiteBurgundy` dans la première ontologie et la classe `BurdeosBlanco` dans la deuxième ontologie. Ce type de correspondance n'est pas encore détecté par la version d'implémentation actuelle des algorithmes ASCO. Avec l'intégration et l'utilisation d'un dictionnaire externe, ces correspondances peuvent bien sûr être trouvées par les algorithmes.

La valeur de f-mesure est maximale à 0,857 et 0,885 pour le seuil de similarité de 0,8 et 0,9 dans ASCO1 et ASCO2, respectivement.

Seuil	F	T	C	I=F-C	M=T-C	Précision P=C/F	Rappel R=C/T	F-mesure FM=2*P*R/(P+R)	Overall O=R*(2-1/P)
0,50	33	33	27	6	6	0.818	0.818	0.818	0.636
0,60	33	33	27	6	6	0.818	0.818	0.818	0.636
0,70	33	33	27	6	6	0.818	0.818	0.818	0.636
0,80	30	33	27	3	6	0.900	0.818	0.857	0.727
0,90	24	33	23	1	10	0.958	0.697	0.807	0.667
0,95	23	33	23	0	10	1.000	0.697	0.821	0.697

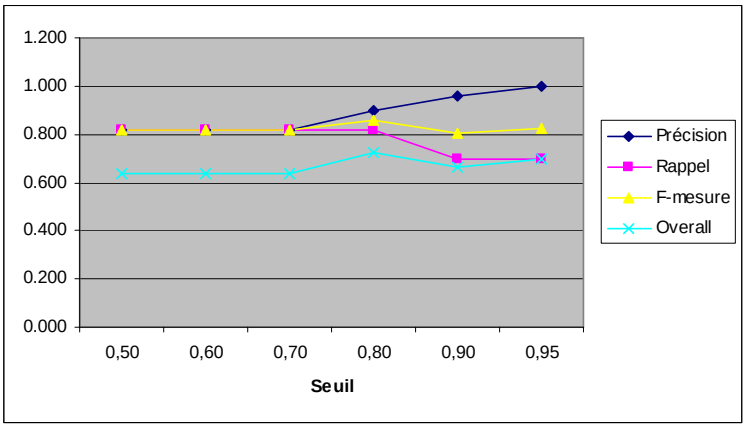


Figure 39 Résultat de ASCO1 sur la paire d'ontologies « Wine »

Seuil	F	T	C	I=F-C	M=T-C	Précision P=C/F	Rappel R=C/T	F-mesure FM=2*P*R/(P+R)	Overall O=R*(2-1/P)
0,70	33	33	29	4	4	0.879	0.879	0.879	0.758
0,75	33	33	29	4	4	0.879	0.879	0.879	0.758
0,80	33	33	29	4	4	0.879	0.879	0.879	0.758
0,85	31	33	28	3	5	0.903	0.848	0.875	0.758
0,90	28	33	27	1	6	0.964	0.818	0.885	0.788
0,95	23	33	23	0	10	1.000	0.697	0.821	0.697

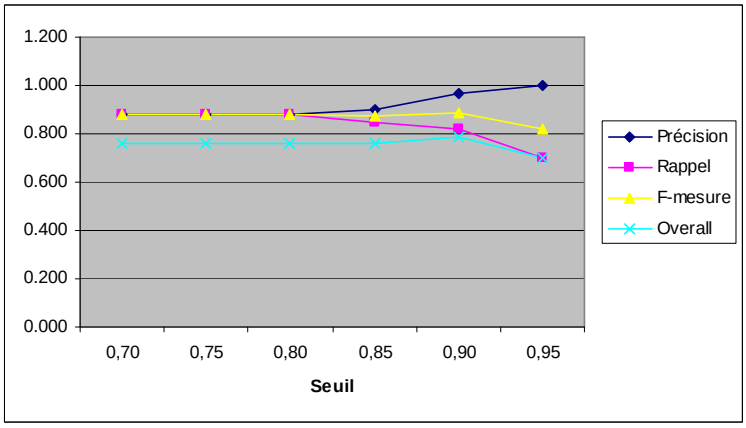


Figure 40 Résultat de ASCO2 sur la paire d'ontologies « Wine »

nRadius_Super nRadius_Sub	F	T	C	I=F-C	M=T-C	Précision P=C/F	Rappel R=C/T	F-mesure FM=2*P*R/(P+R)	Overall O=R*(2-1/P)
0-0	32	33	29	3	4	0.906	0.879	0.892	0.788
2-2	32	33	29	3	4	0.906	0.879	0.892	0.788

Tableau 13 Résultat de ASCO3 sur la paire d'ontologies « Wine »

4.1.3.1.7 La paire d'ontologies « Weapons »

Comme la paire d'ontologies « *Wine* », les ontologies dans cette paire sont en `OWL` et sont stockées en format `N3`. Elles n'ont que des classes (79 classes pour la première ontologie et 81 pour la deuxième), et pas de propriétés ni d'instances. Les classes dans les deux ontologies sont organisées dans des structures assez similaires, et sont nommées de manière similaire. Les résultats renvoyés par les algorithmes ASCO1 et ASCO2 sont donc très bons.

La valeur de f-mesure est maximale à 0,979 et 0,973 pour le seuil de similarité de 0,95 et 0,9 dans ASCO1 et ASCO2, respectivement.

Seuil	F	T	C	I=F-C	M=T-C	Précision P=C/F	Rappel R=C/T	F-mesure FM=2*P*R/(P+R)	Overall O=R*(2-1/P)
0,50	79	73	71	8	2	0.899	0.973	0.934	0.863
0,60	79	73	71	8	2	0.899	0.973	0.934	0.863
0,70	79	73	71	8	2	0.899	0.973	0.934	0.863
0,80	78	73	71	7	2	0.910	0.973	0.940	0.877
0,90	75	73	71	4	2	0.947	0.973	0.959	0.918
0,95	72	73	71	1	2	0.986	0.973	0.979	0.959

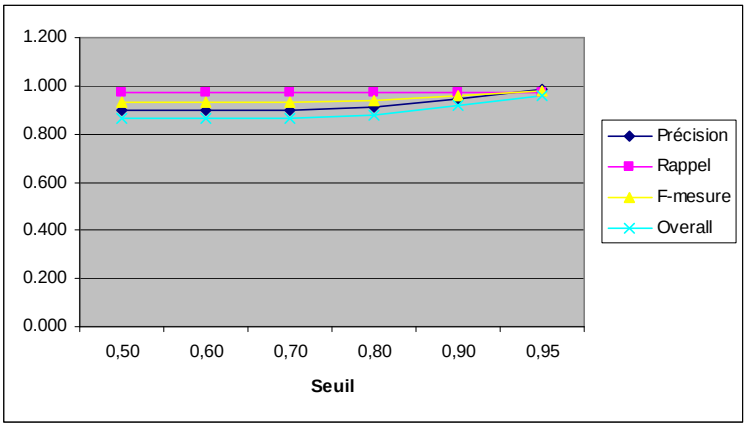


Figure 41 Résultat de ASCO1 sur la paire d'ontologies « Weapons »

Seuil	F	T	C	I=F-C	M=T-C	Précision P=C/F	Rappel R=C/T	F-mesure FM=2*P*R/(P+R)	Overall O=R*(2-1/P)
0,70	79	73	72	7	1	0.911	0.986	0.947	0.890
0,75	79	73	72	7	1	0.911	0.986	0.947	0.890
0,80	79	73	72	7	1	0.911	0.986	0.947	0.890
0,85	77	73	72	5	1	0.935	0.986	0.960	0.918
0,90	73	73	71	2	2	0.973	0.973	0.973	0.945
0,95	70	73	69	1	4	0.986	0.945	0.965	0.932

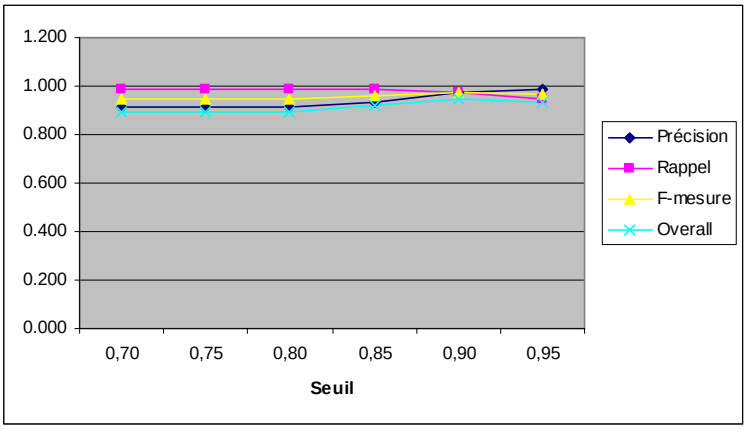


Figure 42 Résultat de ASCO2 sur la paire d'ontologies « Weapons »

4.1.3.1.8 Conclusion

Dans cette section, nous avons présenté les évaluations des trois algorithmes proposés ASCO1, ASCO2 et ASCO3 sur l'ensemble de paires d'ontologies de test fournies dans le cadre de la campagne de tests I^{3}CON. Les 8 paires d'ontologies de test ont été fournies aux deux algorithmes ASCO1 et ASCO2, pour les exécuter avec 6 seuils de similarité différents, soit au total de 96 exécutions. En raison des limites dans la version d'implémentation actuelle de l'application ASCO, qui n'est pas encore bien optimisée, l'algorithme ASCO3 ne fonctionne qu'avec de petites ontologies, qui ont moins d'une trentaine d'entités (dans cette campagne, il y a 4 telles ontologies). De même, cette version courante n'emploie pas encore des dictionnaires externes des synonymes et ceux de langue naturelle, les résultats obtenus sont encore limités. Cependant, les mesures de performance des algorithmes via les mesures telles que la précision, le rappel, la f-mesure, l'overall montrent que les résultats renvoyés par les algorithmes proposés sont de qualité.

Dans la *Figure 43* et la *Figure 44*, nous comparons les résultats obtenus et présentés dans cette section avec le résultat des organisations participant à la campagne I^{3}CON[1] : Lockheed Martin ATL, EXMO/INRIA, Teknowledge, AT&T et University of Karlsruhe. La valeur de f-mesure choisie pour chaque test est la valeur meilleure (la plus élevée) parmi les valeurs obtenues selon les seuils de similarité différents (*Tableau 14*).

[1] http://www.atl.lmco.com/projects/ontology/papers/I3CON-Results.pdf

f-mesure	Animals	Hotels	Computer Networks	Pets	Pets (no instances)	Russia	Wine	Weapons
ASCO1	1.000	0.813	0.769	0.955	0.935	0.807	0.857	0.979
ASCO2	1.000	0.706	0.929	0.949	0.935	0.842	0.885	0.973
ASCO3	1.000	0.813	0.933	-	-	-	0.892	-

Tableau 14 Valeur de f-mesure pour chaque paire d'ontologie et algorithme

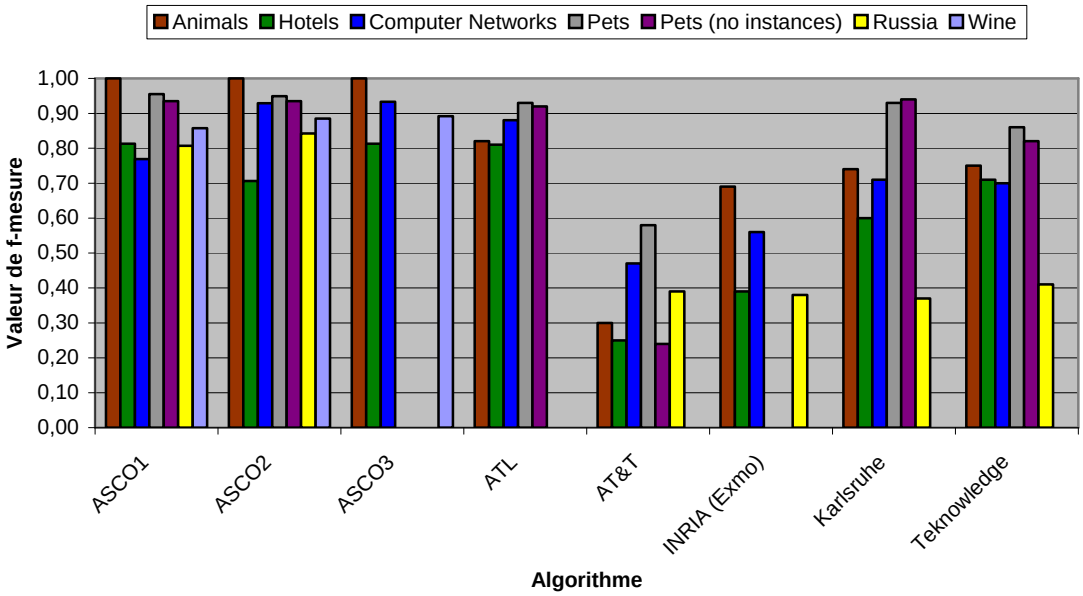


Figure 43 Valeur de f-mesure pour les paires d'ontologies de test et les algorithmes

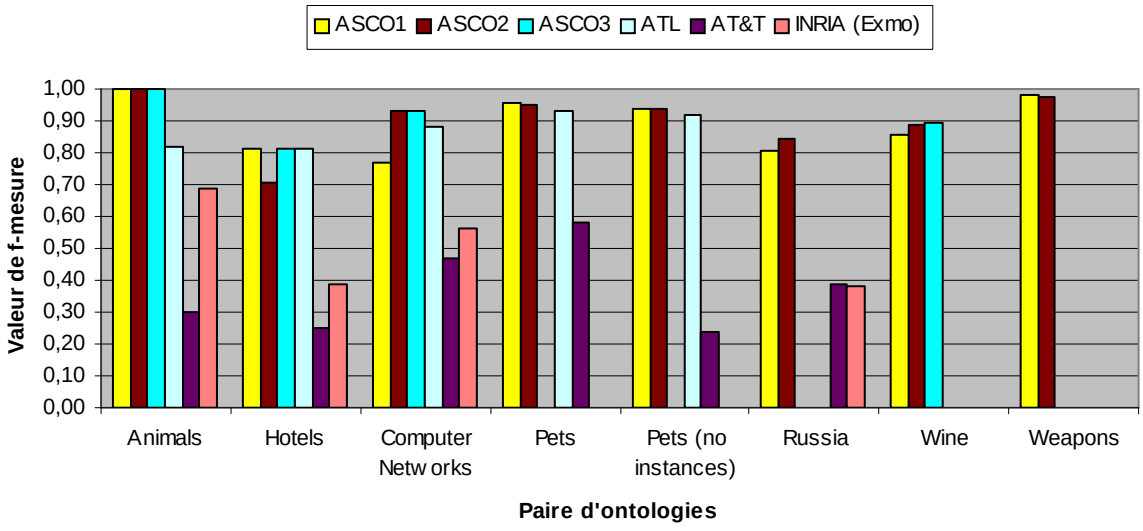


Figure 44 Valeur de f-mesure des algorithmes et les paires d'ontologies de test

En examinant les figures, nous voyons que la paire d'ontologies de test « Russia » est une paire difficile. La meilleure valeur de f-mesure des organisations participant à la campagne est environ de 0,4. Tandis que nos algorithmes, cette valeur est au moins de 0,8. L'algorithme ASCO2 prend en compte les informations de la structure des ontologies, il atteint de la valeur de 0,84. Pour la paire d'ontologies « Animals », tous les trois algorithmes ASCO atteignent la valeur maximale de 1,0. En général, l'algorithme ASCO3 renvoie le meilleur résultat mais son temps d'exécution deviendra très important s'il met en correspondance de grandes ontologies. Il faudrait l'optimiser pour exploiter sa performance pour des paires d'ontologies expressives.

4.1.3.2 Évaluation des paires d'ontologies dans la campagne OAEI

OAEI (Ontology Alignment Evaluation Initiative)[1] est le deuxième effort dans la communauté de recherche de l'intégration d'ontologies et de schémas, afin d'établir un consensus pour l'évaluation des méthodes dans ce domaine. En 2004, après la proposition de I^{3}CON, un concours d'alignement d'ontologies est proposé au 3ème atelier EON (Evaluation of Ontology-based Tools)[2] qui a eu lieu lors de la conférence internationale ISWC (International Semantic Web Conference)[3], appelé aussi le concours d'alignement d'ontologies EON. En 2005, les fondateurs continuent à poursuivre cette initiative et lui donnent un nouveau nom OAEI (Ontology Alignment Evaluation Initiative) et l'accueillent à l'adresse http://oaei.ontologymatching.org. OAEI 2005 a eu lieu à l'atelier Integrating Ontologies[4], lors de la conférence K-Cap 2005 (Third International Conference on Knowledge Capture)[5].

La campagne de tests de OAEI 2005 se compose de 51 tests (51 paires d'ontologies de test). Les tests dans cette suite sont générés à partir d'une onotogie de référence dans le domaine de la bibliographie. Les ontologies dans les paires de test sont décrites en OWL DL et sérialisées dans le format RDF/XML. L'ontologie de référence contient 33 classes nommées, 24 propriétés d'objet, 40 propriétés de données, 56 individus nommés et 20 individus anonymes, soit au total 173 entités.

Les paires d'ontologies de test sont systématiquement générées à partir de l'ontologie de référence en modifiant ou enlevant un certain nombre d'informations afin d'évaluer comment les algorithmes se comportent quand ces informations sont modifiées ou enlevées. Par exemple, les noms d'entité peuvent être remplacés par les chaînes des caractères aléatoires, par des synonymes, ou être traduits en une autre langue ; les commentaires peuvent être supprimés ou changés ; les structures des hiérarchies de classes ou de propriétés peuvent être modifiées…

---

[1] http://oaei.ontologymatching.org

[2] http://km.aifb.uni-karlsruhe.de/ws/eon2004

[3] http://iswc2005.semanticweb.org/

[4] http://km.aifb.uni-karlsruhe.de/ws/intont2005

[5] http://www.kcap05.org/

Nous avons testé les algorithmes ASCO1 et ASCO2 sur les paires d'ontologies de test dans cette campagne, et comparé les résultats obtenus avec ceux des participants publiés à http://oaei.ontologymatching.org/2005/results/. Pour chaque paire d'ontologies de test, deux algorithmes sont exécutés avec 15 seuils de similarité différents (de 0,3 à 1,0 avec le pas d'incrément de 0,05). Cela fait au total de 1530 exécutions. En raison de l'espace, nous ne présentons ici, dans le *Tableau 15*, que les valeurs de précision, de rappel et de f-mesure calculées par nos algorithmes et en comparaison avec celles des autres participants de la campagne OAEI 2005 (EDNA, Falcon – Université de Nanjin, FOAM – Université de Karlsruhe, ctxMatch2-1 – IRST Trento, dublin20 – UC. Dublin, CMS – Université de Southampton, OMAP – CNR/Pisa et OLA – Université de Montréal et INRIA). EDNA (Edit Distance Name Alignment) est un algorithme d'alignement d'ontologies de référence très simple, qui fonctionne par la comparaison des valeurs de similarité des étiquettes des entités en utilisant la mesure de *distance d'édition*. Comme on pouvait le prévoir, les résultats de l'algorithme ASCO1 sur les tests 20X ne sont pas très bons car les paires d'ontologies dans ces tests ont des entités nommées de façons très différentes. Les noms des entités de la deuxième ontologie dans le test 201 sont systématiquement remplacés par des chaînes des caractères aléatoires mais les commentaires sont toujours gardés. Le test 202 va un peu plus loin avec la suppression des commentaires. Les autres tests modifient d'autres informations linguistiques décrivant les entités : par exemple, les noms sont remplacés par des synonymes (test 205), sont traduits en autre langue (test 206)… L'algorithme ASCO1 se base principalement sur la similarité linguistique et la version d'implémentation actuelle de l'ASCO n'emploie pas encore de dictionnaires externes de synonymes (tels que WordNet), de dictionnaires de langue, donc la performance de ASCO1 pour ces tests est faible : la valeur de f-mesure pour le test 201 est de 0,67 ; pour 202, elle est plus faible, à 0,02 ; pour 205 (synonymes), elle est de 0,76 ; pour 206 (traduction), elle est de 0,86 ; pour 209 (synonymes et pas de commentaires), elle est de 0,39. L'algorithme ASCO2 emploie en plus, à part des informations linguistiques des entités, les informations structurelles et la méthode d'agrégation dynamique (les poids pondérés variables), le résultat renvoyé par ASCO2 est amélioré : les valeurs de f-mesure sont de 0,87 0,3 0,92 0,93 0,57 pour les tests 201, 202, 205, 206, 209, respectivement. Les tests 25X et 26X sont les modifications des tests 20X, donc, les résultats des ASCO1 et ASCO2 ne sont pas très bons non plus.

Comme nous en avons discuté dans la section *4.1.3.1.4* et *3.3.1.3*, la version d'implémentation présente de l'application ASCO ne permet pas d'évaluer l'algorithme ASCO3 avec des ontologies ayant plus d'une trentaine d'entités du fait de la complexité de l'algorithme et des problèmes d'optimisation de code Java. L'algorithme ASCO3 n'a donc pas été appliqué pour les tests dans cette campagne OAEI.

Algo	refalign		EDNA		Falcon		FOAM		ctxMatch21		Dublin20		CMS		OMAP		OLA		ASCO1		ASCO2	
Test	*Pré*	*Rap*	*Pré*	*Rap*	*Pré*	*Rap*	*Pré*	*Rap*	*Pré*	*Rap*	*Pré*	*Rap*	*Pré*	*Rap*	*Pré*	*Rap*	*Pré*	*Rap*	*Pré*	*Rap*	*Pré*	*Rap*
101	1.00	1.00	0.96	1.00	1.00	1.00	n/a	n/a	0.87	0.34	1.00	0.99	n/a	n/a	0.96	1.00	1.00	1.00	1.00	1.00	1.00	1.00
103	1.00	1.00	0.96	1.00	1.00	1.00	0.98	0.98	0.87	0.34	1.00	0.99	0.67	0.25	0.96	1.00	1.00	1.00	1.00	1.00	1.00	1.00
104	1.00	1.00	0.96	1.00	1.00	1.00	0.98	0.98	0.87	0.34	1.00	0.99	0.80	0.34	0.96	1.00	1.00	1.00	1.00	1.00	1.00	1.00
201	1.00	1.00	0.03	0.03	0.98	0.98	n/a	n/a	0.00	0.00	0.96	0.96	1.00	0.07	0.80	0.38	0.71	0.62	0.72	0.62	0.87	0.87
202	1.00	1.00	0.03	0.03	0.87	0.87	0.79	0.52	0.00	0.00	0.75	0.28	0.25	0.01	0.82	0.24	0.66	0.56	0.03	0.02	0.31	0.30
203	1.00	1.00	0.96	1.00	1.00	1.00	1.00	1.00	0.87	0.34	1.00	0.99	1.00	0.24	0.96	1.00	1.00	1.00	0.99	0.99	1.00	1.00
204	1.00	1.00	0.90	0.94	1.00	1.00	1.00	0.97	0.87	0.28	0.98	0.98	1.00	0.24	0.93	0.89	0.94	0.94	1.00	1.00	1.00	1.00
205	1.00	1.00	0.34	0.35	0.88	0.87	0.89	0.73	0.36	0.04	0.98	0.97	1.00	0.09	0.58	0.66	0.43	0.42	0.90	0.66	1.00	0.85
206	1.00	1.00	0.51	0.54	1.00	0.99	1.00	0.82	0.30	0.03	0.96	0.95	1.00	0.09	0.74	0.49	0.94	0.93	0.90	0.82	0.99	0.88
207	1.00	1.00	0.51	0.54	1.00	0.99	0.96	0.78	0.30	0.03	0.96	0.95	1.00	0.09	0.74	0.49	0.95	0.94	0.90	0.82	0.99	0.88
208	1.00	1.00	0.90	0.94	1.00	1.00	0.96	0.89	0.87	0.28	0.99	0.96	1.00	0.19	0.96	0.90	0.94	0.94	0.97	0.92	0.98	0.98
209	1.00	1.00	0.35	0.36	0.86	0.86	0.78	0.58	0.36	0.04	0.68	0.56	1.00	0.04	0.41	0.60	0.43	0.42	0.58	0.29	0.57	0.57
210	1.00	1.00	0.51	0.54	0.97	0.96	0.87	0.64	0.40	0.04	0.96	0.82	0.82	0.09	0.88	0.39	0.95	0.94	0.74	0.54	0.74	0.72
221	1.00	1.00	0.96	1.00	1.00	1.00	1.00	1.00	0.87	0.34	1.00	0.99	1.00	0.27	0.96	1.00	1.00	1.00	1.00	1.00	1.00	1.00
222	1.00	1.00	0.91	0.99	1.00	1.00	0.98	0.98	0.73	0.26	1.00	0.99	1.00	0.23	0.96	1.00	1.00	1.00	1.00	1.00	1.00	1.00
223	1.00	1.00	0.96	1.00	1.00	1.00	0.99	0.98	0.83	0.31	0.99	0.98	0.96	0.26	0.96	1.00	1.00	1.00	1.00	1.00	0.98	0.98
224	1.00	1.00	0.96	1.00	1.00	1.00	1.00	0.99	0.87	0.34	1.00	0.99	1.00	0.27	0.96	1.00	1.00	1.00	1.00	1.00	1.00	1.00
225	1.00	1.00	0.96	1.00	1.00	1.00	0.00	0.00	0.84	0.32	1.00	0.99	0.74	0.26	0.96	1.00	1.00	1.00	1.00	1.00	1.00	1.00
228	1.00	1.00	0.38	1.00	1.00	1.00	1.00	1.00	0.92	1.00	1.00	1.00	0.74	0.76	0.92	1.00	1.00	1.00	0.97	1.00	0.97	1.00
230	1.00	1.00	0.71	1.00	0.94	1.00	0.94	1.00	0.83	0.35	0.95	0.99	1.00	0.26	0.89	1.00	0.95	0.97	0.99	0.99	1.00	1.00
231	1.00	1.00	0.96	1.00	1.00	1.00	0.98	0.98	0.87	0.34	1.00	0.99	1.00	0.27	0.96	1.00	1.00	1.00	1.00	1.00	1.00	1.00
232	1.00	1.00	0.96	1.00	1.00	1.00	1.00	0.99	0.84	0.32	1.00	0.99	1.00	0.27	0.96	1.00	1.00	1.00	1.00	1.00	1.00	1.00
233	1.00	1.00	0.38	1.00	1.00	1.00	1.00	1.00	0.55	0.52	1.00	1.00	0.81	0.76	0.92	1.00	1.00	1.00	0.97	1.00	0.97	1.00
236	1.00	1.00	0.38	1.00	1.00	1.00	1.00	1.00	0.86	0.94	1.00	1.00	0.74	0.76	0.92	1.00	1.00	1.00	0.97	1.00	0.97	1.00
237	1.00	1.00	0.91	0.99	1.00	1.00	1.00	0.99	0.79	0.29	1.00	0.99	1.00	0.23	0.95	1.00	0.97	0.98	1.00	1.00	1.00	1.00

238	1.00	1.00	0.96	1.00	0.99	0.99	1.00	0.99	0.76	0.29	0.99	0.98	0.96	0.26	0.96	1.00	0.99	0.99	1.00	1.00	0.98	0.98
239	1.00	1.00	0.28	1.00	0.97	1.00	0.97	1.00	0.74	0.79	0.97	1.00	0.71	0.76	0.85	1.00	0.97	1.00	0.97	1.00	0.97	1.00
240	1.00	1.00	0.33	1.00	0.97	1.00	0.94	0.97	0.78	0.85	0.94	0.97	0.71	0.73	0.87	1.00	0.97	1.00	0.89	1.00	0.97	0.94
241	1.00	1.00	0.38	1.00	1.00	1.00	1.00	1.00	0.58	0.58	1.00	1.00	0.81	0.76	0.92	1.00	1.00	1.00	0.97	1.00	0.97	1.00
246	1.00	1.00	0.28	1.00	0.97	1.00	0.97	1.00	0.75	0.83	0.97	1.00	0.71	0.76	0.85	1.00	0.97	1.00	0.97	1.00	0.97	1.00
247	1.00	1.00	0.33	1.00	0.94	0.97	0.94	0.97	0.77	0.82	0.94	0.97	0.71	0.73	0.87	1.00	0.97	1.00	0.89	1.00	0.97	0.94
248	1.00	1.00	0.06	0.06	0.84	0.82	0.89	0.51	0.00	0.00	0.71	0.25	0.25	0.01	0.82	0.24	0.59	0.46	0.03	0.02	0.30	0.29
249	1.00	1.00	0.04	0.04	0.86	0.86	0.80	0.51	0.00	0.00	0.74	0.29	0.25	0.01	0.81	0.23	0.59	0.46	0.03	0.02	0.32	0.31
250	1.00	1.00	0.01	0.03	0.77	0.70	1.00	0.55	0.00	0.00	1.00	0.09	0.00	0.00	0.05	0.45	0.30	0.24	0.02	0.03	0.13	0.18
251	1.00	1.00	0.01	0.01	0.69	0.69	0.90	0.41	0.00	0.00	0.79	0.32	0.25	0.01	0.82	0.25	0.42	0.30	0.03	0.02	0.44	0.34
252	1.00	1.00	0.01	0.01	0.67	0.67	0.67	0.35	0.00	0.00	0.57	0.22	0.25	0.01	0.82	0.24	0.59	0.52	0.03	0.02	0.26	0.22
253	1.00	1.00	0.05	0.05	0.86	0.85	0.80	0.40	0.00	0.00	0.76	0.27	0.25	0.01	0.81	0.23	0.56	0.41	0.03	0.02	0.42	0.30
254	1.00	1.00	0.02	0.06	1.00	0.27	0.78	0.21	0.00	0.00	NaN	0.00	0.00	0.00	0.03	1.00	0.04	0.03	0.01	0.03	0.19	0.21
257	1.00	1.00	0.01	0.03	0.70	0.64	1.00	0.64	0.00	0.00	1.00	0.09	0.00	0.00	0.05	0.45	0.25	0.21	0.02	0.03	0.13	0.18
258	1.00	1.00	0.01	0.01	0.70	0.70	0.88	0.39	0.00	0.00	0.79	0.32	0.25	0.01	0.82	0.25	0.49	0.35	0.03	0.02	0.36	0.37
259	1.00	1.00	0.01	0.01	0.68	0.68	0.61	0.34	0.00	0.00	0.59	0.21	0.25	0.01	0.82	0.24	0.58	0.47	0.03	0.02	0.38	0.32
260	1.00	1.00	0.00	0.00	0.52	0.48	0.75	0.31	0.00	0.00	0.75	0.10	0.00	0.00	0.05	0.86	0.26	0.17	0.02	0.03	0.22	0.34
261	1.00	1.00	0.00	0.00	0.50	0.48	0.63	0.30	0.00	0.00	0.33	0.06	0.00	0.00	0.01	0.15	0.14	0.09	0.02	0.03	0.20	0.30
262	1.00	1.00	0.01	0.03	0.89	0.24	0.78	0.21	0.00	0.00	NaN	0.00	0.00	0.00	0.03	1.00	0.20	0.06	0.01	0.03	0.14	0.15
265	1.00	1.00	0.00	0.00	0.48	0.45	0.75	0.31	0.00	0.00	0.75	0.10	0.00	0.00	0.05	0.86	0.22	0.14	0.02	0.03	0.33	0.17
266	1.00	1.00	0.00	0.00	0.50	0.48	0.67	0.36	0.00	0.00	0.33	0.06	0.00	0.00	0.01	0.15	0.14	0.09	0.02	0.03	0.18	0.27
301	1.00	1.00	0.48	0.79	0.96	0.80	0.83	0.31	0.00	0.00	0.74	0.64	1.00	0.13	0.94	0.25	0.42	0.38	0.86	0.72	0.92	0.57
302	1.00	1.00	0.31	0.65	0.97	0.67	0.97	0.65	0.00	0.00	0.62	0.48	1.00	0.17	1.00	0.58	0.37	0.33	0.76	0.58	0.74	0.35
303	1.00	1.00	0.40	0.82	0.80	0.82	0.89	0.80	0.32	0.14	0.51	0.53	1.00	0.18	0.93	0.80	0.41	0.49	0.81	0.78	0.51	0.76
304	1.00	1.00	0.71	0.95	0.97	0.96	0.95	0.96	0.58	0.09	0.75	0.70	0.85	0.22	0.91	0.91	0.74	0.66	0.95	0.92	0.90	0.87

Tableau 15 Valeur de *précision* (Pré) et *rappel* (Rap) de différents algorithmes pour chaque test

Le *Tableau 16* résume les valeurs moyennes de la f-mesure pour chaque groupe de test, ainsi que celles de toute la campagne. Ces valeurs sont représentées dans la *Figure 45*. Nous pouvons voir que l'algorithme ASCO2 renvoie en général les résultats meilleurs que l'algorithme ASCO1, surtout dans les cas où des informations linguistiques des entités sont totalement différentes mais les structures des ontologies sont un peu similaires (les tests 20X, 25X, 26X). Par contre, pour les tests avec les ontologies de la bibliographie 301 (BibTeX/MIT), 302 (BibTeX/UMBC), 303 (Karlsruhe), 304 (INRIA), où les concepts sont assez similaires mais organisés et définis différemment, ASCO1 montre des résultats meilleurs que ASCO2.

f-mesure	EDNA	Falcon	FOAM	ctxMatch21	Dublin20	CMS	OMAP	OLA	ASCO1	ASCO2
1XX	0.98	1.00	0.98	0.49	0.99	0.42	0.98	1.00	1.00	1.00
20X	0.52	0.95	0.83	0.21	0.87	0.20	0.65	0.78	0.71	0.82
22X	0.90	1.00	0.99	0.54	0.99	0.46	0.98	1.00	1.00	0.99
23X	0.78	0.99	0.99	0.56	0.99	0.54	0.96	0.99	0.99	0.99
24X	0.35	0.94	0.86	0.74	0.78	0.50	0.74	0.83	0.65	0.75
25X	0.02	0.68	0.54	-	0.32	0.02	0.26	0.35	0.02	0.27
26X	0.02	0.46	0.42	-	0.14	-	0.06	0.14	0.02	0.22
30X	0.59	0.86	0.76	0.18	0.62	0.29	0.72	0.47	0.79	0.67
XXX	0.52	0.86	0.80	0.45	0.71	0.35	0.67	0.70	0.65	0.71

Tableau 16 Valeur moyenne de f-mesure pour chaque groupe de test

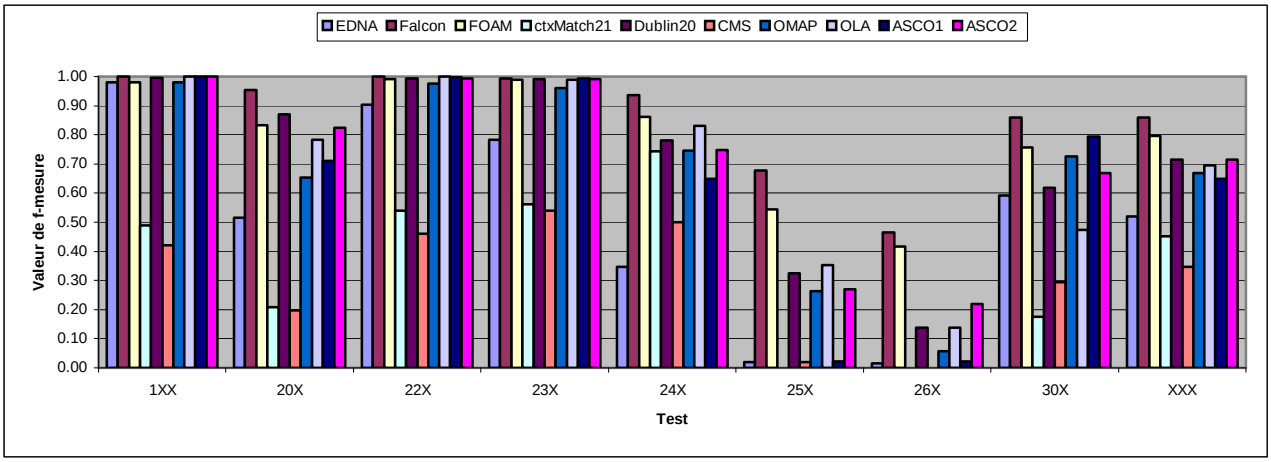


Figure 45 Valeur de f-mesure moyenne des algorithmes et les tests

4.1.3.3 Évaluation des paires d'ontologies réelles et de test

4.1.3.3.1 Évaluation de paire d'ontologies réelles

Les algorithmes ASCO1 et ASCO2 ont été expérimentés avec deux ontologies réelles en RDF(S): O'COMMA, qui a 472 classes et 75 propriétés; et O'Aprobatiom, qui a 460 classes et 92 propriétés. O'COMMA est une ontologie de mémoire d'entreprise, qui a été développée pour le projet européen IST CoMMA [Gandon *et al.*, 2002]. O'Aprobatiom est une ontologie pour le projet dans le domaine de bâtiment, développée dans le cadre d'une coopération entre l'équipe ACACIA (INRIA) et le Centre Scientifique et Technique du Bâtiment (CSTB)[1]. Les deux ontologies ont été développées dans deux projets séparés par différentes personnes, mais elles partagent quelques parties communes, concernant le domaine de bâtiment, donc les résultats obtenus (*Figure 46* et *Figure 47*) sont assez bons : la valeur de la f-mesure est maximale à 0,834 et 0,872 pour le seuil de similarité de 0,7 et 0,75 dans ASCO1 et ASCO2, respectivement.. Les correspondances dans le fichier de référence sont détectées manuellement. Elles se composent de 78 correspondances de classes et 6 correspondances de propriétés, soit 84 correspondances correctes.

Notons que les noms des entités dans l'ontologie O'COMMA sont encodés en anglais et ceux dans l'ontologie O'Aprobatiom sont en français. Par contre, les étiquettes des entités existent dans les deux langues. Par exemple, la classe NonFerroMetalTopic de l'ontologie O'COMMA correspond à la classe MetalNonFerreux de l'ontologie O'Aprobatiom. La plupart des correspondances correctes mais ne pouvant pas être trouvées par les algorithmes quand le seuil de similarité est petit (0,6-0,7) sont des correspondances de propriété. La raison est que les propriétés des deux ontologies sont largement différentes. Il n'y a que 6 correspondances de propriétés correctes sur 75 propriétés de l'ontologie O'COMMA et 92 propriétés de l'ontologie O'Aprobatiom. En plus, les structures des hiérarchies de propriétés de deux ontologies sont aussi très différentes. Les propriétés sont nommées différemment, par exemple FamilyName et NomPersonne. En continuant à augmenter le seuil, nous perdons aussi des correspondances de classe correctes, mais diminuons le nombre des correspondances incorrectes.

Seuil	F	T	C	I=F-C	M=T-C	Précision P=C/F	Rappel R=C/T	F-mesure FM=2*P*R/(P+R)	Overall O=R*(2-1/P)
0,6	175	84	73	102	11	0.417	0.869	0.564	-0.345
0,65	93	84	69	24	15	0.742	0.821	0.780	0.536
0,7	67	84	63	4	21	0.940	0.750	0.834	0.702
0,75	47	84	46	1	38	0.979	0.548	0.702	0.536
0,8	38	84	38	0	46	1.000	0.452	0.623	0.452

[1] http://www.cstb.fr/

0,85	18	84	18	0	66	1.000	0.214	0.353	0.214
0,9	2	84	2	0	82	1.000	0.024	0.047	0.024
0,95	0	84	0	0	84	#DIV/0!	0,000	#DIV/0!	#DIV/0!

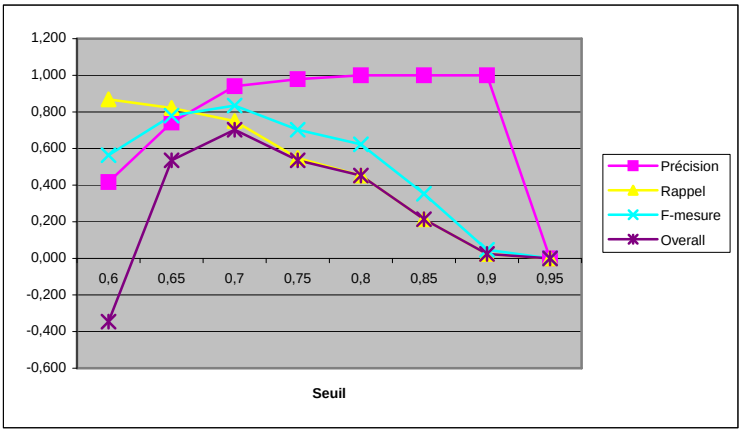


Figure 46 Résultat de ASCO1 sur deux ontologies O'COMMA et O'Aprobatiom

Seuil	F	T	C	I=F-C	M=T-C	Précision P=C/F	Rappel R=C/T	F-mesure FM=2*P*R/(P+R)	Overall O=R*(2-1/P)
0,6	190	84	73	117	11	0.384	0.869	0.533	-0.524
0,65	83	84	70	13	14	0.843	0.833	0.838	0.679
0,7	73	84	68	5	16	0.932	0.810	0.866	0.750
0,75	72	84	68	4	16	0.944	0.810	0.872	0.762
0,8	66	84	63	3	21	0.955	0.750	0.840	0.714
0,85	53	84	53	0	31	1.000	0.631	0.774	0.631
0,9	31	84	31	0	53	1.000	0.369	0.539	0.369
0,95	1	84	1	0	83	1.000	0.012	0.024	0.012

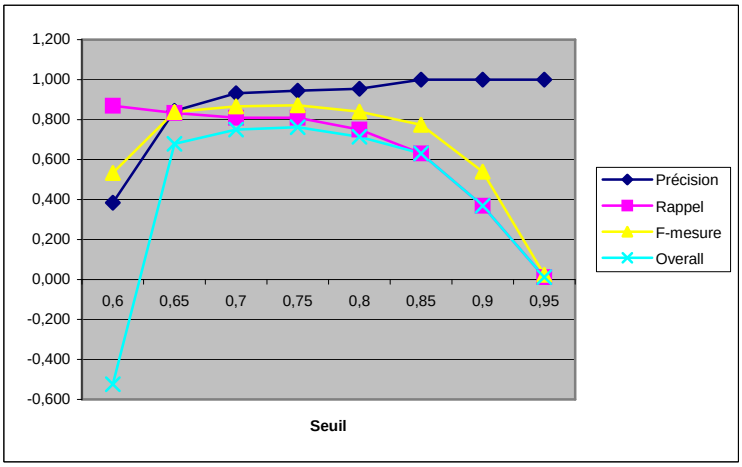


Figure 47 Résultat de ASCO2 sur deux ontologies O'COMMA et O'Aprobatiom

4.1.3.3.2 Évaluation de paire d'ontologies de test

Nous avons créé deux petites ontologies de test, nommées PB1 et PB2, pour examiner les comportements des algorithmes proposés. Les noms des entités et les relations entre elles sont pris de WordNet. Le *Tableau 17* montre les sérialisations en OWL de deux ontologies, et les représentations des O-Graphes (*Définition 39*) sont dans la *Figure 48* avec les liens d'équivalence établis manuellement entre deux entités de deux ontologies.

Ontologie PB1

```
<!DOCTYPE owl [
   <!ENTITY onto1    "http://www-
sop.inria.fr/acacia/ontologies/onto1.owl#">
   <!ENTITY owl
"http://www.w3.org/2002/07/owl#"> ]>

<rdf:RDF
   xmlns:owl
="http://www.w3.org/2002/07/owl#"
   xmlns:rdf ="http://www.w3.org/1999/02/22-
rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-
schema#"
   xml:base  ="&onto1;" >

<owl:Class rdf:ID="Person" />

<owl:Class rdf:ID="Adult">
  <rdfs:subClassOf rdf:resource="#Person"/>
</owl:Class>

<owl:Class rdf:ID="Adult_Female">
  <rdfs:label>Adult Female</rdfs:label>
  <rdfs:label>Woman</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Adult"/>
</owl:Class>

<owl:Class rdf:ID="Man" />

<owl:Class rdf:ID="Adult_Male">
  <rdfs:subClassOf rdf:resource="#Person"/>
  <owl:equivalentClass rdf:resource="#Man"/>
</owl:Class>

<owl:Class rdf:ID="Book">
  <rdfs:subClassOf
rdf:resource="#Publication"/>
  <owl:disjointWith
rdf:resource="#Magazine"/>
</owl:Class>

<owl:Class rdf:ID="Magazine">
  <rdfs:subClassOf
rdf:resource="#Publication"/>
</owl:Class>

<owl:Class rdf:ID="Publication" />

<owl:ObjectProperty rdf:ID="write">
  <rdfs:domain rdf:resource="#Person"/>
  <rdfs:range rdf:resource="#Publication"/>
</owl:ObjectProperty>

</rdf:RDF>
```

Ontologie PB2

```
<!DOCTYPE owl [
   <!ENTITY onto2    "http://www-
sop.inria.fr/acacia/ontologies/onto2.owl#">
   <!ENTITY owl
"http://www.w3.org/2002/07/owl#"> ]>

<rdf:RDF
   xmlns:owl
="http://www.w3.org/2002/07/owl#"
   xmlns:rdf ="http://www.w3.org/1999/02/22-
rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-
schema#"
   xml:base  ="&onto2;" >

<owl:Class rdf:ID="Human" />

<owl:Class rdf:ID="Woman">
  <rdfs:label>Woman</rdfs:label>
  <rdfs:label>Adult Female</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Human"/>
</owl:Class>

<owl:Class rdf:ID="Man">
  <rdfs:subClassOf rdf:resource="#Human"/>
</owl:Class>

<owl:Class rdf:ID="Booklet" />

<owl:ObjectProperty rdf:ID="write">
  <rdfs:subPropertyOf
rdf:resource="#create"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="create">
  <rdfs:domain rdf:resource="#Human"/>
  <rdfs:range rdf:resource="#Booklet"/>
</owl:ObjectProperty>

</rdf:RDF>
```

Tableau 17 Sérialisations en OWL de deux ontologies PB1 et PB2

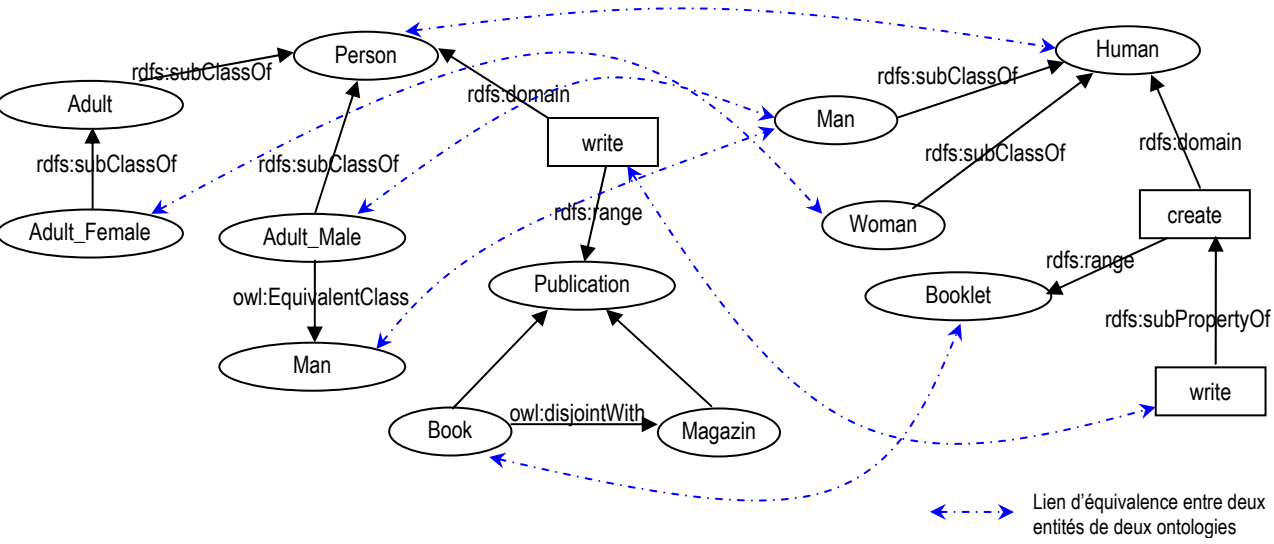


Figure 48 Représentations de O-Graphe pour deux ontologies PB1 et PB2

Les entités correspondantes trouvées avec leurs valeurs de similarité totales calculées par les algorithmes ASCO1 et ASCO2 sont montrées dans le *Tableau 18*.

Ordre	Entité de première ontologie	Entité de deuxième ontologie	Valeur de similarité
1	Adult_Male	Man	0.920
2	write	write	0.920
3	Man	Man	0.920
4	Book	Booklet	0.874
5	Magazine	Man	0.835
6	Publication	Booklet	0.746
7	Adult	Man	0.713
8	Person	Human	0.699
9	Adult_Female	Woman	0.623

Ordre	Entité de première ontologie	Entité de deuxième ontologie	Valeur de similarité
1	Man	Human	0.673
2	write	create	0.757
3	Adult_Male	Human	0.673
4	Magazine	Man	0.780
5	Publication	Booklet	0.768
6	Adult	Man	0.725
7	Person	Human	0.709
8	Adult_Female	Woman	0.657
9	Book	Booklet	0.549

Tableau 18 Les correspondances et les valeurs de similarité renvoyées par ASCO1 (à gauche) et ASCO2 (à droite)

Les valeurs de f-mesure, de précision, de rappel et d'overall des algorithmes ASCO1 et ASCO2 pour les différents seuils de similarité sont dans la *Figure 49* et la *Figure 50*, respectivement.

Seuil	F	T	C	I=F-C	M=T-C	Précision P=C/F	Rappel R=C/T	F-mesure FM=2*P*R/(P+R)	Overall O=R*(2-1/P)
0.55	9	6	6	3	0	0.667	1.000	0.800	0.500
0.60	9	6	6	3	0	0.667	1.000	0.800	0.500
0.65	8	6	5	3	1	0.625	0.833	0.714	0.333
0.70	7	6	4	3	2	0.571	0.667	0.615	0.167
0.75	5	6	4	1	2	0.800	0.667	0.727	0.500
0.80	5	6	4	1	2	0.800	0.667	0.727	0.500

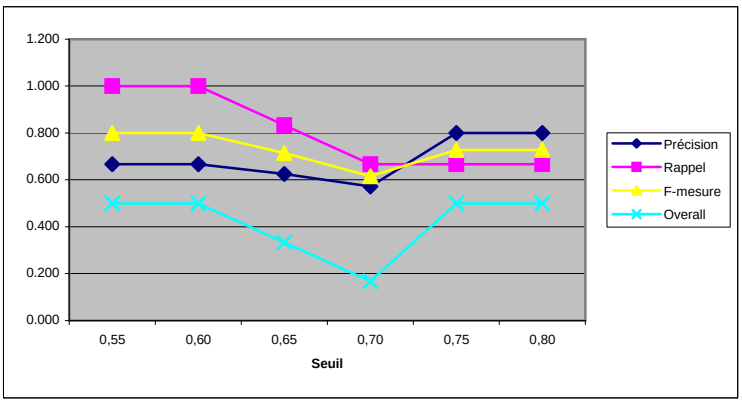


Figure 49 Résultat de ASCO1 sur la paire d'ontologies PB1 et PB2

Seuil	F	T	C	I=F-C	M=T-C	Précision P=C/F	Rappel R=C/T	F-mesure FM=2*P*R/(P+R)	Overall O=R*(2-1/P)
0.45	9	6	4	5	2	0.444	0.667	0.533	-0.167
0.50	9	6	4	5	2	0.444	0.667	0.533	-0.167
0.55	8	6	3	5	3	0.375	0.500	0.429	-0.333
0.60	8	6	3	5	3	0.375	0.500	0.429	-0.333
0.65	7	6	2	5	4	0.286	0.333	0.308	-0.500
0.70	5	6	1	4	5	0.200	0.167	0.182	-0.500

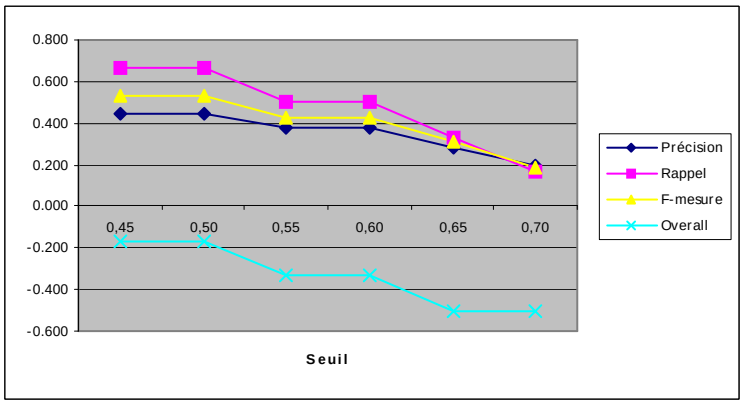


Figure 50 Résultat de ASCO2 sur la paire d'ontologies PB1 et PB2

Nous pouvons voir que la valeur de f-mesure est maximale à 0,8 et 0,533 pour le seuil de similarité de 0,6 et 0,5 dans ASCO1 et ASCO2, respectivement. L'algorithme ASCO1 exploite principalement les informations linguistiques dans les définitions des entités et quelques informations structurelles (des similarités des voisines dans les hiérarchies de classes et de propriétés) pour calculer la similarité de deux entités. Il peut donc renvoyer correctement les bonnes correspondances telles que `write` – `write` (avec la valeur de similarité de 0,92) ou `Book` – `Booklet` (0,874). La valeur de similarité de `Person` – `Human` est assez élevée car les valeurs de similarité de leurs voisines sont aussi élevées. Pour le cas de `Adult_Female` – `Woman` (0,623), ces entités ont deux étiquettes similaires : « Adult Female » et « Woman ». Cependant, ASCO1 détecte incorrectement trois paires d'entités `Magazine` – `Man` (0,835), `Publication` – `Booklet` (0,746) et `Adult` – `Man` (0,713) si les seuils de similarité sont moins de 0,713. En augmentant le seuil de similarité, il peut éliminer ces correspondances incorrectes mais il enlève aussi les autres bonnes correspondances telles que `Person` – `Human` (0.699) ou `Adult_Female` – `Woman` (0.623). C'est pour cela que la valeur de f-mesure maximale de ASCO1 n'atteint pas 1,0.

L'algorithme ASCO2 se base principalement sur la structure de deux ontologies, spécialement sur les structures locales des entités. Dans cet exemple, les deux ontologies sont assez différentes au niveau de leurs structures, et donc, le résultat obtenu par ASCO2 est faible. Il détecte de manière erronée, par exemple, que `Man` – `Human` est une correspondance correcte car leur valeur de similarité calculée est de 0,673, qui est plus élevée que la valeur de similarité de la paire `Man` – `Man` (0,577).

Contrairement à ASCO1 et ASCO2, dans cet exemple, l'algorithme ASCO3 montre une performance impressionnante avec l'introduction de la notion de « *saut* ». Dans le cas de la recherche la clique maximale dans le graphe d'association de deux O-Graphes (*Figure 48*) représentés deux ontologies, sans introduction de la notion de « *saut* » (les paramètres de l'algorithme ASCO3 `nRadius_Super` et `nRadius_Sub` sont égaux à 0), le résultat obtenu est très faible avec le sous-graphe commun maximal de deux O-Graphes contenant 5 paires de nœuds incorrectes et une seule paire correcte (*Figure 51*). Rappelons qu'un nœud du graphe d'association se compose de deux nœuds de O-Graphe, qui correspondent à deux entités de deux ontologies. Alors, les nœuds dans la clique maximale trouvée dans le graphe d'association correspondent aux paires des entités des deux ontologies, qui sont considérées par ASCO3 comme des correspondances correctes entre deux ontologies.

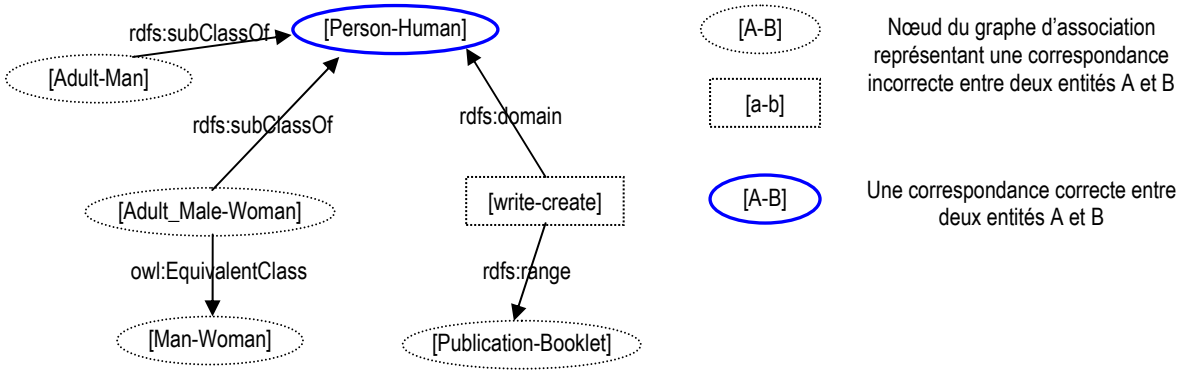


Figure 51 Le sous-graphe commun maximal de deux O-Graphes dans le cas où il n'y a pas de « *saut* » (`nRadius_Super` = 0 et `nRadius_Sub` = 0)

Avec l'introduction de la notion de « *saut* », les nœuds et les arcs du graphe d'association sont créés en intégrant la sémantique des primitives de `OWL`. Si le paramètre `nRadius_Super` est égal à `n`, les nœuds de O-Graphe dans le rayon de `n`, via les liens de subsomption entre des entités, à partir du nœud en question, sont aussi pris en compte dans la création des nœuds et des arcs du graphe d'association. Autrement dit, les ancêtres d'un nœud de O-Graphe, jusqu'à la `n`-ième génération précédente, sont aussi pris en compte. Par exemple, dans le O-Graphe correspondant à la première ontologie (*Figure 48*), les ancêtres de `Adult`, dans le rayon de 1 (`nRadius_Super` = 1), est `Person` ; les ancêtres de `Adult_Female`, dans le rayon de 2 (`nRadius_Super` = 2), sont `Adult` et `Person`. Alors, si l'arc [`Adult_Female-Man`] – `rdfs:subClassOf` – [`Adult-Human`] est créé dans le graphe d'association, l'arc [`Adult_Female-Man`] – `rdfs:subClassOf` – [`Person-Human`] est aussi créé (dans le cas `nRadius_Super` = 1), car `Person` est dans le rayon de 1 à partir du nœud `Adult`. Cela signifie que si `Adult_Female-Man` est une bonne correspondance, alors soit `Adult` – `Human`, soit `Person` – `Human` est aussi une bonne correspondance.

De même, `nRadius_Sub` permet de prendre en compte des nœuds descendants d'un nœud dans le rayon de `nRadius_Sub`. Notons que la recherche des nœuds est effectuée grâce aux liens de subsomption entre les entités. Autrement dit, la recherche se fait dans les hiérarchies de classes ou de relations. Par exemple, avec `nRadius_Sub` = 1, si l'arc [`Publication-Booklet`] – `rdfs:range` – [`write-create`] est créé dans le graphe d'association, les arcs suivants [`Publication-Booklet`] – `rdfs:range` – [`write-write`] (`write` est dans le rayon de 1 à partir de `create`), [`Book-Booklet`] – `rdfs:range` – [`write-create`] et [`Magazine-Booklet`] – `rdfs:range` – [`write-create`] (`Book` et `Magazine` sont dans le rayon de 1 à partir de `Publication`) sont aussi créés.

Les figures suivantes (*Figure 52*, *Figure 53* et *Figure 54*) montrent les différents résultats renvoyés par l'algorithme ASCO3, qui sont des cliques maximales du graphe d'association ou les sous-graphes communs maximaux de deux O-Graphes, correspondant aux valeurs différentes de `nRadius_Super` et `nRadius_Sub`. Nous voyons qu'aux valeurs de 1 pour les deux paramètres (ou le « *saut* » de 1), nous obtenons le résultat parfait.

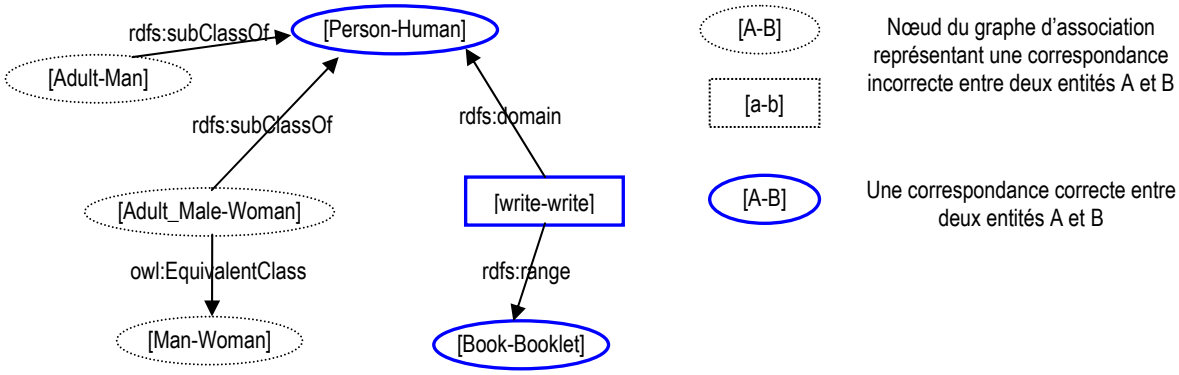


Figure 52 Le sous-graphe commun maximal de deux O-Graphes dans le cas où il y a le « *saut en haut* » de 1 (`nRadius_Super` = 1 et `nRadius_Sub` = 0)

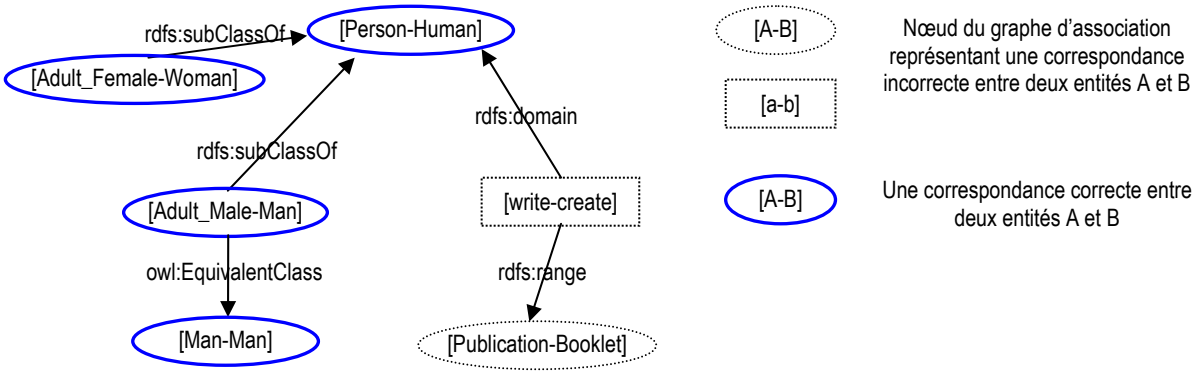


Figure 53 Le sous-graphe commun maximal de deux O-Graphes dans le cas où il y a le « *saut en bas* » de 1 (`nRadius_Super` = 0 et `nRadius_Sub` = 1)

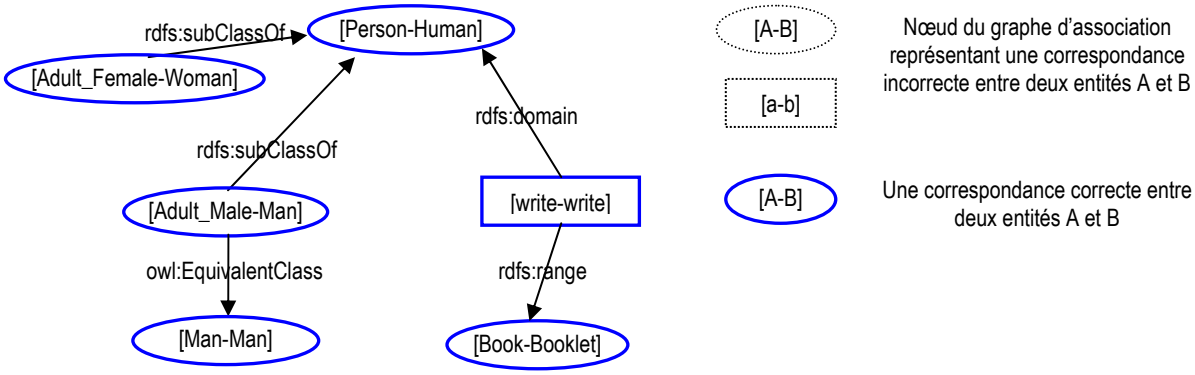


Figure 54 Le sous-graphe commun maximal de deux O-Graphes dans le cas où il y a les « *sauts dans les deux sens* » de 1 (`nRadius_Super`=1 et `nRadius_Sub`= 1)

Les valeurs de f-mesure, de précision, de rappel et d'overall de l'algorithme ASCO3 pour les différentes valeurs de `nRadius_Super` et `nRadius_Sub` sont montrées dans la *Figure 55*.

Seuil	F	T	C	I=F-C	M=T-C	Précision P=C/F	Rappel R=C/T	F-mesure FM=2*P*R/(P+R)	Overall O=R*(2-1/P)
0 - 0	6	6	1	5	5	0.167	0.167	0.167	-0.667
1 - 0	6	6	3	3	3	0.500	0.500	0.500	0.000
0 - 1	6	6	4	2	2	0.667	0.667	0.667	0.333
1 - 1	6	6	6	0	0	1.000	1.000	1.000	1.000
2 - 1	6	6	6	0	0	1.000	1.000	1.000	1.000
2 - 2	6	6	6	0	0	1.000	1.000	1.000	1.000

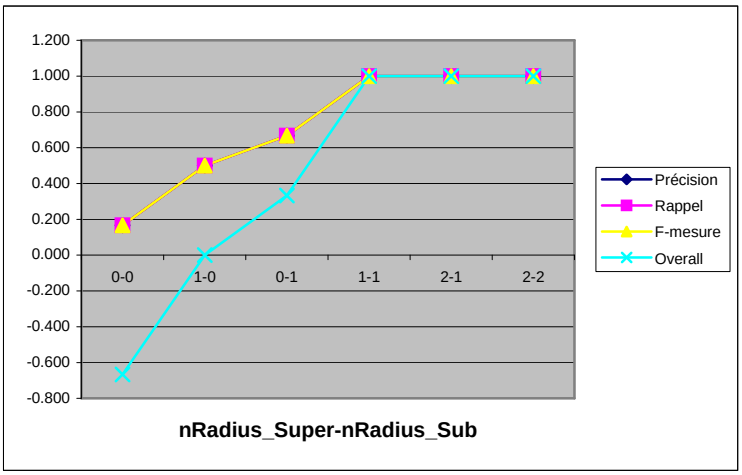


Figure 55 Résultat de ASCO3 sur la paire d'ontologies PB1 et PB2

4.2 Conclusion

Dans ce chapitre, nous avons présenté les expérimentations de nos trois algorithmes d'alignement d'ontologies.

Les expérimentations sont effectuées en trois parties : deux campagnes d'évaluation des algorithmes d'alignement I^{3}CON et OAEI, et une partie avec des ontologies réelles. Malgré des limites dans la version d'implémentation actuelle en Java des algorithmes (l'application ASCO), (il n'y a pas d'utilisation de WordNet via des APIs en Java, pas d'utilisation de dictionnaires externes de langue, l'implémentation n'est pas encore bien optimisée) les algorithmes montrent des résultats probants pour les tests avec 10 paires d'ontologies différentes en obtenant des valeurs de f-mesure plus élevées que celles fournies par les autres participants (*Figure 43* et *Figure 44*), surtout avec la paire d'ontologies très difficiles à aligner « *Russia* », les algorithmes ASCO1 et ASCO2 renvoient les valeurs de f-mesure deux fois plus élevées que celles des autres participants (0,8 contre 0,4).

Pour la deuxième campagne d'évaluation OAEI, où 51 paires d'ontologies de test sont générées systématiquement et artificiellement par les changements d'une ou plusieurs des informations décrites des entités de deux ontologies de référence dans le domaine de la bibliographie, nos algorithmes n'arrivent pas à la première place. Une des raisons est que les correspondances correctes fournies dans les fichiers de référence ne sont pas très fiables. Par exemple, dans les fichiers de référence fournis dans la campagne OAEI, nous pouvons trouver la correspondance entre la propriété `http://oaei.inrialpes.fr/2005/benchmarks/101/onto.rdf#lccn`, qui est une `owl:DatatypeProperty` dans la première ontologie 101, et la propriété `http://oaei.inrialpes.fr/2005/benchmarks/304/onto.rdf#lccn`, qui est une `owl:ObjectProperty` dans la deuxième ontologie 304. Les algorithmes ASCO sont conçus pour faire aligner les ontologies en `OWL DL/Lite`, ils ne peuvent pas détecter un tel type de relations entre des entités, qui n'est pas autorisé en `OWL DL/Lite` (`owl:DatatypeProperty` et `owl:ObjectProperty` sont disjoints en `OWL DL/Lite`).

Les résultats de ASCO1 et ASCO2 obtenus dans le test avec deux ontologies réelles O'COMMA et O'Aprobatiom sont aussi encourageants : les valeurs de f-mesure dans le meilleur cas dépassent 0,83. L'implémentation courante en Java de l'algorithme ASCO3 qui n'est pas bien optimisée et n'intègre pas des heuristiques pour réduire le temps d'exécution, ne permet pas d'évaluer des tests avec des ontologies ayant plus d'une trentaine d'entités. Ces limites sont étudiées et améliorées dans les versions futures de ASCO.

La dernière paire d'ontologies de test montre le point fort de l'algorithme ASCO3 avec l'importance de la notion de « *saut* ». Dans ce test, lorsque les deux autres algorithmes ASCO1 et ASCO2 ne peuvent pas renvoyer de résultat parfait, ASCO3 nous donne le résultat correct et parfait en employant les « *sauts* ». La sémantique des primitives de `OWL` est bien exploitée via cette notion de « *saut* ».

En général, les résultats obtenus par l'algorithme ASCO2 sont meilleurs que ceux renvoyés par ASCO1, et ASCO3 montre les meilleurs résultats. Dans certains cas, si les ontologies à aligner sont plus similaires au niveau linguistique qu'au niveau structurel, ASCO1 est un meilleur choix. Pour les cas contraires, si la structure des ontologies sont proches, ASCO2 et ASCO3 sont plus efficaces.

L'alignement d'ontologies facilite des tâches d'échange des informations dans l'environnement hétérogène du Web sémantique, dans le cas où les ontologies sont déjà *existantes*.

Toujours pour le but de s'adapter et de co-exister avec l'hétérogénéité du Web sémantique, nous proposons dans le chapitre suivant, un modèle de représentation des connaissances multi-points de vue, le *modèle multi-points de vue*, et un langage de représentation pour ce modèle, le langage `MVP-OWL`. Le modèle et le langage proposés permettent de construire de *nouvelles* ontologies, mais en prenant en compte différents points de vue. Les ontologies construites selon le modèle et le langage proposés sont nommées les *ontologies multi-points de vue*. Telles ontologies facilitent des échanges des informations dans un cadre d'un Web sémantique d'entreprise en harmonisant le consensus et l'hétérogénéité dans telle organisation.

Chapitre 5

Ontologie multi-points de vue

Dans ce chapitre nous proposons notre approche pour la représentation d'ontologies multi-points de vue en étendant le langage d'ontologie OWL. Tout d'abord, nous allons proposer notre définition de la notion de point de vue et expliciter les objectifs de travail que nous visons par rapport à cette définition. Puis, nous présentons le modèle multi-points de vue MVP permettant de représenter des ontologies avec multi-points de vue. Ensuite, une extension du langage d'ontologie de OWL est présentée : elle permet d'exprimer le modèle MVP dans un langage d'ontologie étendu de OWL : MVP-OWL. Enfin, nous présentons une méthode permettant de construire des ontologies multi-points de vue dans le langage MVP-OWL à partir des ontologies en OWL en exploitant les résultats de nos travaux sur l'alignement d'ontologies. Des opérateurs dédiés à la gestion des ontologies multi-points de vue sont également étudiés.

5.1 Définition de travail

Dans la section *1.3*, nous avons analysé la notion de point de vue et quelques travaux exploitant la notion de point de vue pour représenter des connaissances. Dans le cadre du Web sémantique, l'ontologie est utilisée pour représenter des connaissances d'un domaine de discours. Notre objectif est d'offrir une représentation des connaissances exploitant la notion de point de vue dans le cadre du Web sémantique : représentation des ontologies en prenant en compte la notion de point de vue.

Il est nécessaire d'atteindre un certain niveau de consensus pour que les connaissances représentées dans une ontologie puissent être utilisées, partagées, échangées efficacement. Une ontologie correspond à une modélisation du monde dans un domaine de discours. Un tel consensus dans une ontologie requiert idéalement que les membres de la communauté concernée s'engagent à utiliser cette ontologie, et reposent ainsi sur une modélisation commune du monde réel. Cependant, si l'on passe d'une petite communauté des utilisateurs à une bien plus grande communauté, ou bien si l'on considère une organisation telle qu'une entreprise, voire une entreprise virtuelle constituée de plusieurs organisations en collaboration, la maintenance d'un tel consensus et d'une modélisation commune du monde devient très difficile. On a besoin de pouvoir exprimer le consensus dans un contexte particulier pour chaque groupe ou communauté, et de décomposer le monde selon les vues des groupes ou communautés différents : c'est là qu'intervient la notion de point de vue.

Dans le Web sémantique, une ontologie, en tant que modèle de connaissance pour un domaine de discours, devrait être donc construite dans un environnement multi-points de vue, de manière à prendre en compte la diversité des sources de connaissances et les différentes catégories d'utilisateurs, mais tout en gardant un certain niveau de consensus. Elle devrait permettre de faire cohabiter à la fois l'hétérogénéité et le consensus des agents humains dans le domaine. Nous appelons une ontologie construite de telle manière, une *ontologie multi-points de vue*.

Un point de vue correspond à un contexte ou à une situation, où des connaissances à propos d'un objet, d'un concept ou d'une entité sont exprimées et considérées comme valides et vraies selon ce point de vue. Un point de vue correspond aussi à une vue, où l'on examine des caractéristiques d'un concept ou d'une entité, qui sont considérées comme des caractéristiques pertinentes du concept ou de l'entité dans cette vue. Un point de vue peut provenir d'une personne, d'un groupe de personnes, d'une communauté dans une entreprise ou une organisation. On peut toujours atteindre un certain niveau de consensus entre les gens dans un groupe, dans une organisation, contrairement à l'Internet, où l'hétérogénéité est vaste et incontrôlable.

De ces observations ci-dessus, en inspirant et en étendant de la définition de point de vue de Ribière [Ribière, 1999], notre définition de travail sera donc : « *Un point de vue est l'interface permettant (1) l'accès à un sous-ensemble d'informations ou de connaissances pertinentes et (2) l'interprétation contextuelle des éléments de connaissances* ».

Nous visons donc à construire et exploiter un *Web sémantique* dans une *organisation hétérogène*, comportant différentes sources de connaissances et différentes catégories d'utilisateurs, en construisant une *ontologie multi-points de vue* permettant d'exprimer des terminologies, des engagements ontologiques et des connaissances : une telle ontologie doit ainsi traduire l'hétérogénéité, et exprimer à la fois les points de vue des personnes de l'organisation, et le consensus entre les membres de l'organisation.

5.2 Modèle multi-points de vue MVP

Notre modèle de connaissance multi-points de vue est proposé dans le contexte de travail indiqué dans la section *5.1*. Dans ce cadre, nous faisons les hypothèses suivantes :

- Les entités dans le modèle sont référencées par des URIs.
- Dans un modèle, une entité est soit une classe (un concept), une propriété (une relation), une instance (un individu) ou un point de vue.
- L'interprétation des définitions des classes, des propriétés et des points de vue dans un modèle doit être cohérente et non contradictoire. Par exemple, si nous utilisons le terme `Person` pour désigner la classe correspondant à toutes les personnes dans le monde, alors l'emploi ou la description de ce terme partout dans le modèle doit toujours être une référence à cet ensemble de personnes. Autrement dit, les connaissances au niveau conceptuel dans une organisation, représentées par le modèle de connaissances multi-points de vue, doivent être cohérentes et consensuelles.
- Les connaissances, les descriptions, les définitions concernant un individu, correspondant à un objet réel, *selon* un point de vue sont valides, vraies, cohérentes et non contradictoires dans ce point de vue. Par contre, les descriptions concernant un même individu selon des points de vue différents peuvent éventuellement être incohérentes ou contradictoires. Autrement dit, les connaissances au niveau assertionnel dans une organisation peuvent être cohérentes ou non.

Selon les hypothèses ci-dessus, nous définissons les entités du modèle multi-points de vue `MVP` (*Figure 56*):

Point de vue – Un point de vue est une entité dans le modèle multi-points de vue. Un point de vue correspond à une vue perspective composée des certaines caractéristiques (propriétés ou attributs) d'une classe (concept). Ces caractéristiques sont les caractéristiques pertinentes par rapport à ce point de vue. Un point de vue permet de filtrer des caractéristiques d'une classe. Le point de vue utilisé en combinaison avec la relation de subsomption (spécialisation) entre deux classes permet de contraindre l'interprétation de la sous-classe. Par exemple, si l'on a défini le point de vue de sexe $\texttt{vp}_{\texttt{sex}}$ qui est composé de la caractéristique `sex` d'une classe, alors l'interprétation de la classe `Man`, qui est définie comme sous-classe de la classe `Person` selon le point de vue

vp_{sex}, est la contrainte que toutes les instances de la classe `Man` doivent être les instances de la classe `Person` et que les valeurs de la caractéristique `sexe` de ces instances doivent être la même valeur (qui est `male`). La classe `Man` a bien sûr d'autres caractéristiques, telles que `age`, `size`, `height`,… mais dans le point de vue vp_{sex}, on ne considère que la caractéristique `sex` de la classe `Man`.

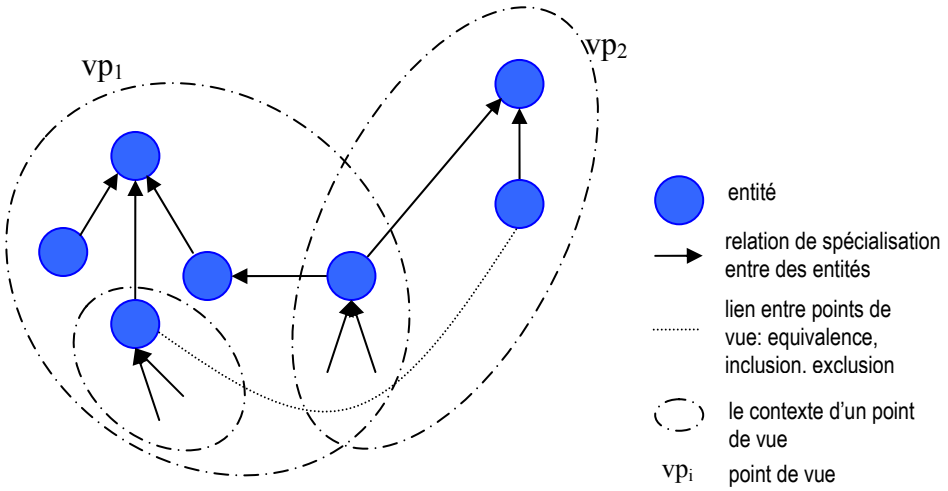


Figure 56 Modèle multi-points de vue MVP

Un point de vue correspond aussi à un contexte particulier où les individus (objets réels) sont décrits et utilisés. Les descriptions à propos d'un individu selon un point de vue sont cohérentes et valides dans le contexte défini par le point de vue.

Nous reprenons et adaptons deux types de points de vue présentés dans [Ribière, 1999] :

- *Point de vue perspective* : Ce sont les points de vue définissant des vues « perspectives » qui indexent des descriptions consensuelles d'un même objet par différents ontologistes ou différents experts. Ces vues sont complémentaires et forment une vision cohérente du monde.

- *Point de vue opinion* : Ce sont les points de vue définissant des vues « opinions » qui indexent des vues non consensuelles de certains ontologistes, experts, annotateurs ou utilisateurs finaux. Ces vues représentent indépendamment les unes des autres des visions incomplètes du monde et peuvent être inconsistantes, non compatibles ou contradictoires.

Le modèle de connaissance multi-points de vue est conçu pour représenter des connaissances dans une entreprise dans le cadre du Web sémantique, il est nécessaire donc que les connaissances au niveau conceptuel ou terminologique soient cohérentes et compatibles. Ainsi, les points de vue utilisés dans les définitions des classes sont du type de point de vue perspective. Par contre, au niveau assertionnel, les connaissances, les énoncés ou les annotations à propos des objets réels, décrits dans des points de vue différents en employant des concepts (classes), des relations (propriétés) consensuelles dans le modèle, peuvent être soit compatibles soit contradictoires. Les points de vue

utilisés pour décrire les objets réels peuvent être donc les points de vue perspectives (si les descriptions sont compatibles) ou les points de vue opinions (si les descriptions sont incompatibles).

Il existe dans le modèle un *point de vue général*, nommé VP_G. Toutes les connaissances, les descriptions, les définitions à propos des entités (sauf les individus) dans le modèle sont considérées valides et vraies selon ce point de vue général. Autrement dit, il existe un consensus et une cohérence au niveau global dans le modèle (au niveau de l'entreprise, qui emploie ce modèle de connaissance multi-points de vue). Tous les autres points de vue, qui sont définis par les ontologistes ou les utilisateurs finaux, sont définis et valables selon ce point de vue général. Les points de vue, les classes, les propriétés dans le modèle sont donc cohérents et ne sont pas contradictoires entre eux. Cela assure le consensus au niveau conceptuel et terminologique dans une entreprise. Le point de vue général est un point de vue perspective.

Notre objectif est de permettre de construire une ontologie multi-points de vue dans le cadre du web sémantique, en nous basant sur les standards du W3C tels que `RDF(S)`, `OWL`. Le modèle multi-points de vue est construit en visant la simplicité et l'utilisabilité pour qu'il puisse être employé par des organisations dans le cadre du Web sémantique. Ainsi, dans le modèle proposé, nous n'avons pas introduit de relations entre les points de vue. Par exemple, un point de vue ne peut pas être défini comme sous-point de vue d'un autre point de vue. Cette capacité pourrait être affectée en considérant comme [Ribière, 1999] qu'un point de vue correspond à un ensemble de critères qui caractérisent le contexte défini par le point de vue, et que l'ajout d'autres critères (caractéristiques) à cet ensemble créera un autre point de vue, qui sera un sous-point de vue du point de vue en question [Ribière, 1999].

Les points de vue, comme les autres entités dans le modèle `MVP`, sont référencés par des URIs. Un même point de vue peut être référencé par deux ou plusieurs URIs, mais un URI ne peut représenter qu'un seul point de vue. Par exemple, si le point de vue de l'équipe ACACIA est référencé par le URI `acacia:vp`, alors, toutes les autres références utilisant `acacia:vp` réfèrent au même point de vue en question : le point de vue de l'équipe ACACIA. Par contre, ce même point de vue pourrait être référencé par un autre URI, par exemple, `inria:vp_acacia`.

Classe – Une classe est une entité du modèle multi-points de vue `MVP`. Une classe correspond à un ensemble d'individus (objets réels), qui ont les mêmes caractéristiques définies par la classe.

Une classe est définie par (1) des annotations et (2) ses liens avec d'autres entités (classes, propriétés, instances) définies dans le même modèle multi-points de vue ou dans un autre modèle multi-points de vue : une classe peut être définie comme l'intersection, l'union d'autres classes ; ou comme le complément, l'équivalence, la disjonction avec une autre classe ; ou l'énumération des instances ; ou la restriction sur certaines propriétés.

Dans le modèle `MVP`, une classe peut être définie comme sous-classe d'une autre classe *selon* un point de vue. Par défaut, toutes les relations de subsomption entre deux classes sans spécifier un point de vue sont considérées comme les relations de

subsomption définies avec le point de vue général VP_G. Si une classe est définie comme sous-classe d'une autre classe selon un point de vue, alors les instances de la classe à définir ne doivent être examinées qu'avec les caractéristiques de la classe mère, qui sont définies dans le point de vue. Rappelons que tous les points de vue utilisés dans la définition de subsomption entre deux classes sont les points de vue perspectives. Cela assure que l'interprétation des classes définies dans le modèle est cohérente et non contradictoire.

Nous admettons le *multi-héritage selon plusieurs points de vue* dans la définition d'une classe : une classe peut être sous-classe directe de plusieurs classes selon plusieurs points de vue différents (*Figure 57*). Par exemple, la classe `Man` est définie comme sous-classe de la classe `Male` selon le point de vue vp_{sex}, et comme sous-classe de la classe `Person` selon le point de vue $vp_{evolution}$. Un tel multi-héritage n'est pas permis dans certains modèles de représentation des connaissances prenant en compte la notion multi-points de vue tels que le modèle CVista [Ribière, 1999]. Aussi, une classe peut avoir plusieurs sous-classes selon plusieurs points de vue.

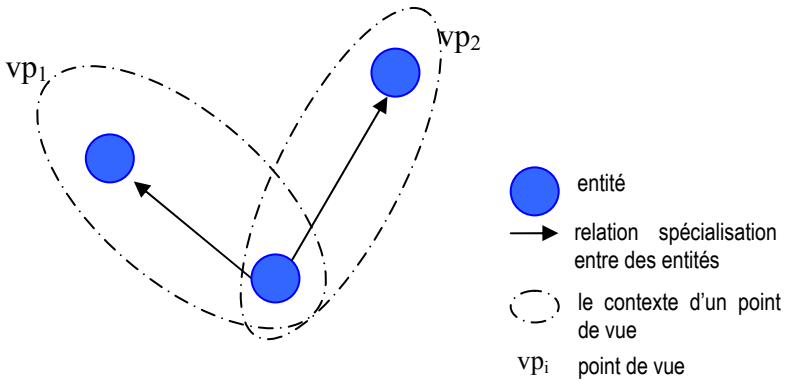


Figure 57 Multi-points de vue et multi-héritage

Le point de vue permet alors d'exprimer les critères sur lesquels, nous nous basons pour décomposer une classe en une ou plusieurs sous-classes.

Propriété (Relation) – Les propriétés sont des entités dans le modèle multi-points de vue `MVP`. Les propriétés permettent d'exprimer des faits généraux à propos des membres des classes ou des faits spécifiques à propos des individus.

Comme les classes, une propriété peut être définie comme sous-propriété d'une autre propriété et le multi-héritage est admis pour les propriétés : une propriété peut être sous-propriété de plusieurs propriétés. Par contre, ces définitions ne sont effectuées que selon le point de vue général.

Liens entre points de vue – On peut définir des passerelles entre des classes définies comme sous-classes d'autres classes selon les points de vue différents, ainsi qu'entre des propriétés et des instances : *équivalence*, *inclusion*, *exclusion* :

- Le lien d'équivalence entre deux classes définies comme sous-classes d'autres classes selon deux points de vue distincts permet d'identifier et d'exprimer ces deux classes ayant le même sens mais utilisées et vues chacune dans un contexte différent, selon un point de vue différent. Le lien d'équivalence permet aussi d'exprimer l'équivalence (avec la même signification) entre des propriétés et entre des instances.

- Le lien d'inclusion entre deux classes définies comme sous-classes d'autres classes selon deux points de vue distincts permet d'identifier et d'exprimer si le sens de la première classe inclut celui de la deuxième classe. Le lien d'inclusion permet aussi d'exprimer l'inclusion entre des propriétés.

- Le lien d'exclusion entre deux classes définies comme sous-classes d'une autre classe selon un même point de vue permet d'identifier les classes pour lesquelles il ne sera pas possible, pour une même instance, d'appartenir en même temps aux deux ensembles d'instances correspondant à ces deux classes.

Les liens entre points de vue permettent de relier les sous-classes décomposées différemment selon des points de vue différents et permettent donc de raisonner avec les classes ainsi que les individus définis dans le modèle en entier, à travers des points de vue différents.

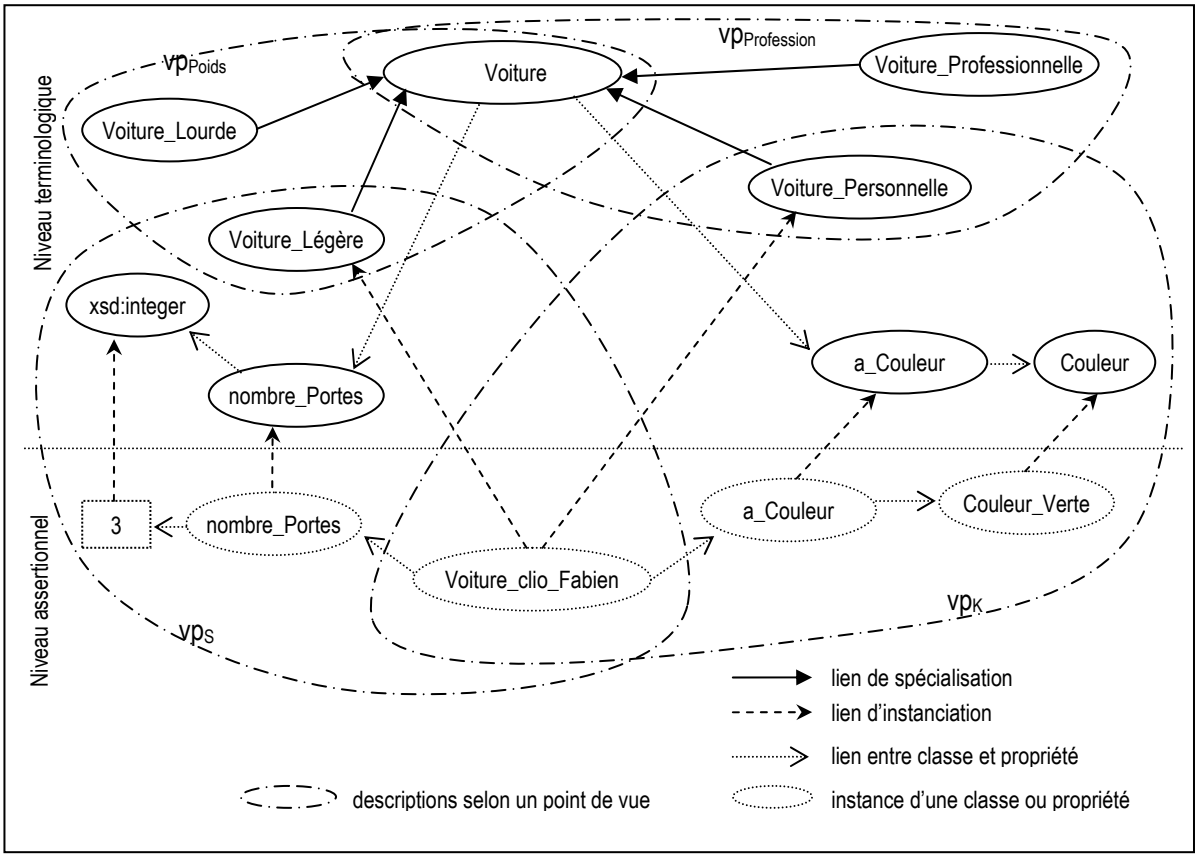


Figure 58 Multi présentation de l'objet réel avec multi-points de vue

Instance (**Individu** ou **Objet réel**) – L'instance d'une classe correspond à un individu (objet) dans le monde réel. Un individu peut être décrit selon un ou plusieurs points de vue (*Figure 58*). Cela permet aussi d'exprimer la multi-représentation d'un objet réel. Par exemple, l'objet réel `voiture_clio_Fabien` est considéré comme une `voiture_légère` d'après le point de vue VP_S, et comme une `voiture_personnelle` selon le point de vue VP_K.

Les descriptions à propos d'un individu sont effectuées dans un ou plusieurs points de vue. Si une description n'est pas associée à un point de vue, nous considérons qu'elle est décrite selon le point de vue général VP_G.

Contrairement aux classes du modèle multi-points de vue `MVP`, où les définitions de subsomption selon plusieurs points de vue doivent être compatibles et non contradictoires, les descriptions différentes à propos d'un objet réel selon des points de vue différents peuvent être incompatibles et contradictoires. Cela permet à différentes personnes, à différents groupes d'exprimer, de décrire un objet réel comme ils le souhaitent et selon leur point de vue. Par contre, les descriptions à propos d'un objet réel effectuées dans un même point de vue doivent être cohérentes et non contradictoires. Les descriptions sont effectuées par des annotations.

Annotation – Les annotations sont les énoncés sur les objets réels. Les annotations peuvent être énoncées selon différents points de vue. Les annotations sont énoncées par des annotateurs ou des utilisateurs finaux et sont donc subjectives et peuvent être contradictoires entre elles. Par exemple, l'objet réel `voiture_clio_Fabien` est considéré comme une voiture, avec la couleur verte selon le point de vue de Sylvain, mais d'après Khaled, elle a une couleur verte claire. Par contre, les annotations énoncées dans un point de vue donné ne peuvent pas être contradictoires. Les annotations sont employées pour décrire des objets réels.

L'annotation est représentée par un quadruplet $Q = <I_S, I_P, I_0, VP>$, comprenant deux instances de classe I_S et I_0, une instance de propriété I_P, et un point de vue `VP`. L'annotation est dite définie (énoncée) selon le point de vue `VP`.

Nous pouvons exprimer que d'après un point de vue, un objet réel est constitué d'instances de plusieurs classes (la multi-représentation). Par exemple, d'après un point de vue, `clio_Fabien` est à la fois une instance de la classe `Voiture_Legere` et une instance de la classe `Voiture_Personnelle` (*Figure 58*). Nous pouvons aussi exprimer que d'après un point de vue, un objet réel a une propriété, et la valeur de cette propriété est soit une instance, soit une valeur littérale. Une même propriété pour un objet réel peut avoir différentes valeurs selon différents points de vue. Par exemple, d'après un point de vue, `clio_Fabien` a 5 portes mais selon un autre point de vue, elle n'a que 4 portes.

Base d'annotations – Une base d'annotations est un ensemble d'annotations énoncées par les annotateurs du système à propos des objets réels dans le domaine d'intérêt d'une entreprise. Les annotateurs peuvent énoncer leur annotations soit dans les points de vue définis dans le modèle multi-points de vue `MVP` utilisé par l'entreprise, soit dans des points de vue définis par eux-mêmes en les ajoutant dans le modèle `MVP` de l'entreprise.

Les connaissances extraites de la base d'annotations peuvent être filtrées selon un point de vue particulier. Les utilisateurs finaux du système peuvent récupérer à partir de la base d'annotations, des connaissances, qui sont pertinentes pour eux, en filtrant la base par un point de vue déjà défini par des ontologistes ou des annotateurs du système. Ils peuvent aussi définir des points de vue eux-mêmes, et déclarer des équivalences entre ces nouveaux points de vue avec les points de vue déjà existants dans le système, ensuite filtrer la base d'annotations selon leurs propres points de vue. D'autre part, les connaissances dans la base d'annotations peuvent être toutes récupérées, sans être filtrées : il suffit de n'indiquer aucun point de vue particulier à filtrer.

5.3 Langage de représentation des ontologies multi-points de vue

Dans cette section, nous présentons notre proposition du langage d'ontologie permettant de représenter une ontologie multi-points de vue dans le cadre du web sémantique. Le langage d'ontologie proposé, nommé `MVP-OWL`, est basé sur le langage d'ontologie `OWL` recommandé par le W3C[1], et est basé sur le modèle multi-points de vue que nous venons de présenter dans la section *5.2*. `MVP-OWL` est une extension de `OWL DL`, par l'ajout de nouvelles primitives (*Tableau 19*) permettant d'exprimer les concepts (classes), les faits, les annotations à propos des objets réels en prenant en compte la notion de point de vue. `MVP-OWL` modifie la sémantique prédéfinie des constructeurs de `OWL DL` pour pouvoir prendre en compte des expressions du point de vue.

Le langage d'ontologie proposé `MVP-OWL` permet de représenter des ontologies multi-points de vue modélisées selon le modèle multi-points de vue présenté dans *5.2*. On peut définir une classe comme sous-classe d'autres classes selon des points de vue différents. Chaque point de vue correspond à un ensemble des critères que les instances d'une classe qui est définie selon ce point de vue doivent satisfaire : dans un point de vue, nous ne considérons que certains critères (attributs) d'une classe. Ces critères sont considérés pertinents et importants dans ce point de vue. Par exemple, si nous considérons une classe avec le point de vue de sexe, qui est caractérisé par le critère « sexe », nous n'examinons que les valeurs de l'attribut « sexe » des instances appartenant à la classe en question. Ces instances ont d'autres attributs mais dans ce point de vue, nous ne nous y intéressons pas.

Nous appelons `vp` le préfixe de l'espace de noms pour les points de vue. Ce nom `vp` correspond à l'espace de noms http://www-sop.inria.fr/ACACIA/ontologies/mvpowl.owl#

```
xmlns:vp = "http://www-sop.inria.fr/ACACIA/ontologies/mvpowl.owl#"
```

[1] http://www.w3.org/TR/2004/REC-owl-ref-20040210/

Nom d'Instance	rdf:type
vp:GeneralViewpoint	vp:PerspectiveViewpoint

Nom de Classe	rdfs:subClassOf
vp:Viewpoint	rdfs:Class
vp:PerspectiveViewpoint	vp:Viewpoint
vp:OpinionViewpoint	vp:Viewpoint
vp:ClassWithViewpoint	owl:Class

Nom de Propriété	Type	Domaine	Co-domaine
vp:belongToViewpoint	owl:ObjectProperty	owl:Thing	vp:Viewpoint
vp:characterisedBy	owl:ObjectProperty	vp:Viewpoint	rdf:Property
vp:onClass	owl:ObjectProperty	vp:ClassWithViewpoint	owl:Class
vp:onViewpoint	owl:ObjectProperty	vp:ClassWithViewpoint	vp:PerspectiveViewpoint
vp:INCL	rdf:Property	-	-
vp:EXCL	rdf:Property	-	-
vp:EQUIV	rdf:Property	-	-

Tableau 19 Nouvelles primitives de MVP-OWL

Tandis que les points de vue associés aux relations de subsomption entre des classes doivent satisfaire la condition de cohérence expliquée dans *5.2* (i.e. il n'y a aucune contradiction dans l'interprétation du modèle représenté par l'ontologie), les interprétations des définitions des individus avec des points de vue peuvent être contradictoires. Nous appelons les points de vue dans le premier cas comme les *points de vue perspectives* et les points de vue pour le deuxième cas, les *points de vue opinions*. Ils sont définis en `MVP-OWL` par deux primitives `vp:PerspectiveViewpoint` et `vp:OpinionViewpoint`, respectivement. Les définitions des classes avec des points de vue sont effectuées à l'aide des primitives `vp:ClassWithViewpoint`, `vp:onClass` et `vp:onViewpoint`.

Nous construisons le langage d'ontologie multi-points de vue `MVP-OWL` afin que les définitions des entités (classes, propriétés, instances) respectent autant que possible la syntaxe de `OWL`, seule la sémantique (l'interprétation) est modifiée (si nécessaire) pour que l'on puisse prendre en compte la notion de point de vue.

En raison de la similarité entre `OWL` et `MVP-OWL`, comme `OWL`, le langage `MVP-OWL` est également présenté en deux parties dans les sous-sections suivantes: la syntaxe abstraite du langage `MVP-OWL` (*5.3.1*) et la sémantique directe du langage `MVP-OWL` selon la théorie des modèles (*5.3.2*). Pour souligner de nouvelles choses ajoutées ou modifiées dans `MVP-OWL` en comparaison avec `OWL`, telles que de nouvelles primitives, contraintes, de nouveaux vocabulaires… nous les représentons en bleu. La *Figure 59* montre le lien de subsomption entre les nouvelles classes de `MVP-OWL` (représentées en bleu) avec les classes de `OWL`.

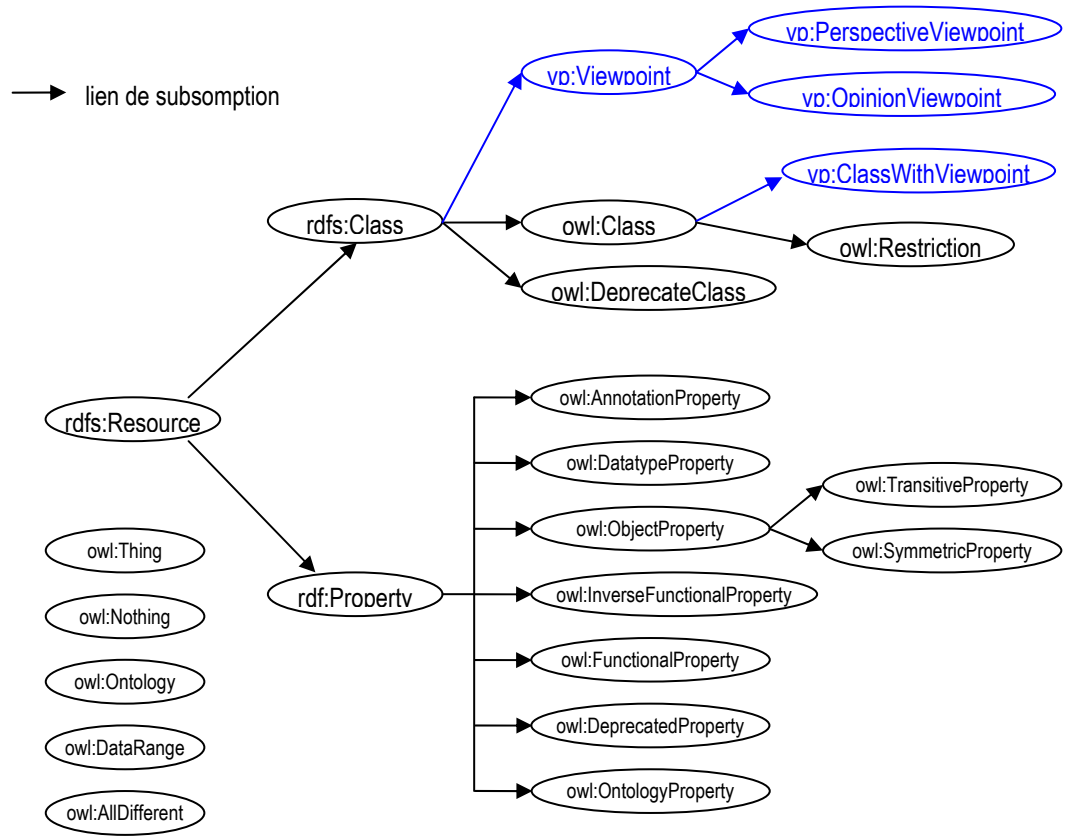


Figure 59 Taxonomie des classes de MVP-OWL

5.3.1 Syntaxe abstraite du langage d'ontologie MVP-OWL

Comme la section 2 dans « OWL Web Ontology Language Semantics and Abstract Syntax »[1] décrivant la syntaxe abstraite du langage d'ontologie `OWL`, la syntaxe abstraite pour `MVP-OWL` est proche de celle des « frames », où les descriptions à propos d'une entité (classe, propriété, instance) sont données dans une construction complexe (composées de plusieurs productions). La syntaxe abstraite est présentée au moyen d'une version de BNF étendue : les terminaux sont entre guillemets, les non-terminaux sont en gras et sans guillemet. Les alternatives sont soit séparées par des barres verticales (|), soit décrivées dans différentes productions (formules). Les composantes qui ne peuvent apparaître au plus qu'une seule fois, sont entre crochets ([...]); les composantes qui peuvent apparaître zéro ou plusieurs fois entre accolades ({...}). Les espaces blancs sont ignorés dans les productions.

En raison de la similarité entre `OWL` et `MVP-OWL`, nous présentons ici les constructeurs de `MVP-OWL` sans les explications s'ils sont repris (similaires avec les

[1] http://www.w3.org/TR/2004/REC-owl-semantics-20040210/syntax.html

homologues) de OWL. Les explications pour tels constructeurs peuvent être consultées à l'adresse du W3C[1]. Par contre, en introduisant les constructeurs que nous proposons concernant la notion de point de vue, nous expliquerons la signification du constructeur.

Ontologies en MVP-OWL

ontology *::= 'Ontology('* ***[ontologyID]*** *{* ***directive*** *} ')'*
directive *::= 'annotation('* ***ontologyPropertyID ontologyID*** *')'*
| *'annotation('* ***annotationPropertyID URIreference*** *')'*
| *'annotation('* ***annotationPropertyID dataLiteral*** *')'*
| *'annotation('* ***annotationPropertyID individual*** *')'*
| ***axiom***
| ***fact***

Identificateurs en MVP-OWL

datatypeID *::=* ***URIreference***
classID *::=* ***URIreference***
individualID *::=* ***URIreference***
ontologyID *::=* ***URIreference***
datavaluedPropertyID *::=* ***URIreference***
individualvaluedPropertyID *::=* ***URIreference***
annotationPropertyID *::=* ***URIreference***
ontologyPropertyID *::=* ***URIreference***
viewpointID *::=* ***perspectiveViewpointID*** | ***opinionViewpointID***
perspectiveViewpointID *::=* ***URIreference***
opinionViewpointID *::=* ***URIreference***
generalViewpoint *::= http://www-sop.inria.fr/ACACIA/ontologies/mvpowl.owl#GeneralViewpoint*

Le modèle multi-points de vue offre la notion de point de vue. Les points de vue doivent être identifiables et pouvoir être référencés. Dans MVP-OWL, les points de vue sont identifiés par des URIs, comme le sont les classes, les propriétés ou les instances. Les points de vue utilisés dans les définitions des classes sont les points de vue perspectives car les interprétations des classes de l'ontologie doivent être cohérentes et non contradictoires. Ils sont représentés par PerspectiveViewpointID. Les points de vue utilisés dans les descriptions des individus sont soit les points de vue perspectives, soit les points de vue opinions, et sont représentés par PerspectiveViewpointID et OpinionViewpointID, respectivement.

Annotations en MVP-OWL

annotation *::= 'annotation('* ***annotationPropertyID URIreference*** *')'*
| *'annotation('* ***annotationPropertyID dataLiteral*** *')'*
| *'annotation('* ***annotationPropertyID individual*** *')'*

Points de vue en MVP-OWL

axiom *::= 'PerspectiveViewpoint('* ***perspectiveViewpointID*** *{* ***annotation*** *}*
{ 'characterisedBy(' ***propertyID*** *')' } ')'*
axiom *::= 'OpinionViewpoint('* ***opinionViewpointID*** *{* ***annotation*** *} { 'characterisedBy('* ***propertyID*** *')' } ')'*

propertyID *::=* ***datavaluedPropertyID***
| ***individualvaluedPropertyID***
| ***annotationPropertyID***

Un point de vue est référencé par un URI et défini par des annotations, en utilisant les propriétés d'annotation de OWL telles que owl:versionInfo, rdfs:label, rdfs:comment, rdfs:seeAlso, et rdfs:isDefinedBy ou des propriétés d'annotation définies par utilisateurs. Un point de vue peut aussi être caractérisé par des

critères (propriétés) en employant la primitive `vp:characterisedBy`. Les propriétés caractérisant un point de vue sont les propriétés pertinentes et importantes dans le point de vue.

Faits en `MVP-OWL`

fact *::=* ***individual***
fact *::= 'SameIndividual('* ***individualID individualID*** *{**individualID**} ')'*
| 'DifferentIndividuals(' ***individualID individualID*** *{**individualID**} ')'*
| 'SameIndividuals(' 'belongToViewpoint(' ***ViewpointID*** *')'* ***individualID individualID*** *{**individualID**} ')'*
| 'DifferentIndividuals(' 'belongToViewpoint(' ***ViewpointID*** *')'* ***individualID individualID*** *{**individualID**} ')'*

individual *::= 'Individual('* *[* ***individualID*** *]* *{* ***annotation*** *}* *{ 'type('* ***type*** *')' }* *{* ***value*** *} ')'*
| 'Individual(' *[* ***individualID*** *]* *'belongToViewpoint('* ***ViewpointID*** *')'*
{ ***annotation*** *}* *{ 'type('* ***type*** *')' }* *{* ***value*** *} ')'*

value *::= 'value('* ***individualvaluedPropertyID individualID*** *')'*
| 'value(' ***individualvaluedPropertyID individual*** *')'*
| 'value(' ***datavaluedPropertyID dataLiteral*** *')'*
type *::=* ***description***
dataLiteral *::=* ***typedLiteral*** *|* ***plainLiteral***
typedLiteral *::=* ***lexicalForm^^URIreference***
plainLiteral *::=* ***lexicalForm*** *|* ***lexicalForm@languageTag***
lexicalForm *::= as in RDF, a unicode string in normal form C*
languageTag *::= as in RDF, an XML language tag*

Les individus sont décrits dans un point de vue ou non. De même, les individus peuvent aussi être déclarés similaires ou différents selon un point de vue ou non. Si un individu n'est pas décrit dans un point de vue, il est considéré comme décrit dans le point de vue général VP_G, qui est un point de vue perspective, c'est-à-dire les descriptions à propos de cet individu sont valides, vraies et non contradictoires dans le modèle en entier. Par contre, les descriptions à propos d'un individu dans un point de vue opinion ne sont valides et pertinentes que dans ce point de vue opinion. Ainsi, les interprétations des descriptions entre des points de vue opinions différents peuvent être incompatibles et contradictoires.

Comme dans `OWL DL`, les individus dans `MVP-OWL` sont aussi typés en utilisant n'importe quel constructeur de description d'une classe ou d'une propriété.

Axiomes des classes et des propriétés de `MVP-OWL`

Dans une ontologie multi-points de vue décrite en `MVP-OWL`, à part des annotations concernant l'ontologie elle-même et des faits (des individus), la partie principale concerne des axiomes fournissant des informations au sujet des classes et des propriétés dans l'ontologie. Le constructeur `classWithViewpoint` permet de décomposer une classe en plusieurs sous-classes selon les critères spécifiés dans un point de vue. L'interprétation (la sémantique) de ce constructeur est donnée formellement dans la section *5.3.2.2*.

classWithViewpoint *::= 'classWithViewpoint(' 'onClass('* ***description*** *')'*
'onViewpoint(' ***perspectiveViewpointID*** *')' ')'*

axiom *::= 'Class('* ***classID*** *['Deprecated']* ***modality*** *{* ***annotation*** *}* *{* ***description*** *}* *')'*
modality *::= 'complete' | 'partial'*

```
axiom ::= 'EnumeratedClass(' classID ['Deprecated'] { annotation } { individualID } ')'

axiom ::= 'DisjointClasses(' description description { description } ')'
      | 'EquivalentClasses(' description { description } ')'
      | 'SubClassOf(' description description ')'
      | 'SubClassOf(' description classWithViewpoint ')'

axiom ::= 'Datatype(' datatypeID ['Deprecated'] { annotation } )'

description ::= classID
         | restriction
         | 'unionOf(' { description } ')'
         | 'intersectionOf(' { description } ')'
         | 'complementOf(' description ')'
         | 'oneOf(' { individualID } ')'

restriction ::= 'restriction(' datavaluedPropertyID dataRestrictionComponent
                              { dataRestrictionComponent } ')'
         | 'restriction(' individualvaluedPropertyID individualRestrictionComponent
                        { individualRestrictionComponent } ')'
dataRestrictionComponent ::= 'allValuesFrom(' dataRange ')'
         | 'someValuesFrom(' dataRange ')'
         | 'value(' dataLiteral ')'
         | cardinality
individualRestrictionComponent ::= 'allValuesFrom(' description ')'
         | 'someValuesFrom(' description ')'
         | 'value(' individualID ')'
         | cardinality
cardinality ::= 'minCardinality(' non-negative-integer ')'
         | 'maxCardinality(' non-negative-integer ')'
         | 'cardinality(' non-negative-integer ')'
dataRange ::= datatypeID | 'rdfs:Literal'
         | 'oneOf(' { dataLiteral } ')'

axiom  ::= 'DatatypeProperty('
             datavaluedPropertyID
             ['Deprecated'] { annotation }
             { 'super(' datavaluedPropertyID ')'} ['Functional']
             { 'domain(' description ')' } { 'range(' dataRange ')' } ')'

axiom ::= 'ObjectProperty('
             individualvaluedPropertyID
             ['Deprecated'] { annotation }
             { 'super(' individualvaluedPropertyID ')' }
             [ 'inverseOf(' individualvaluedPropertyID ')' ] [ 'Symmetric' ]
             [ 'Functional' | 'InverseFunctional' | 'Functional' 'InverseFunctional' | 'Transitive' ]
             { 'domain(' description ')' } { 'range(' description ')' } ')'

axiom ::= 'AnnotationProperty(' annotationPropertyID { annotation } ')'
        | 'OntologyProperty(' ontologyPropertyID { annotation } ')'


axiom ::= 'EquivalentProperties(' datavaluedPropertyID datavaluedPropertyID
             { datavaluedPropertyID } ')'
   | 'SubPropertyOf(' datavaluedPropertyID  datavaluedPropertyID ')'
   | 'EquivalentProperties(' individualvaluedPropertyID individualvaluedPropertyID
                          { individualvaluedPropertyID } ')'
   | 'SubPropertyOf(' individualvaluedPropertyID  individualvaluedPropertyID ')'
```

Liens entre point de vue en MVP-OWL

```
axiom ::= 'EQUIV(' description description { description } ')'
    | 'EQUIV(' datavaluedPropertyID datavaluedPropertyID { datavaluedPropertyID } ')'
    | 'EQUIV(' individualvaluedPropertyID individualvaluedPropertyID
            { individualvaluedPropertyID } ')'
    | 'EQUIV(' annotationPropertyID annotationPropertyID { annotationPropertyID } ')'
    | 'EQUIV(' ontologyPropertyID ontologyPropertyID { ontologyPropertyID } ')'
    | 'EQUIV(' individualID individualID { individualID } ')'

axiom ::= 'INCL(' description description ')'
    | 'INCL(' datavaluedPropertyID datavaluedPropertyID ')'
    | 'INCL(' individualvaluedPropertyID individualvaluedPropertyID ')'
    | 'INCL(' annotationPropertyID annotationPropertyID ')'
    | 'INCL(' ontologyPropertyID ontologyPropertyID ')'

axiom ::= 'EXCL(' description description { description } ')'
```

`MVP-OWL` permet d'exprimer des rapports entre les classes ou les individus définis selon différents points de vue, tels que le rapport d'équivalence, le rapport d'inclusion et le rapport d'exclusion. `MVP-OWL` emploie trois primitives `vp:EQUIV`, `vp:INCL`, `vp:EXCL` pour exprimer ces types de relation, respectivement.

5.3.2 Sémantique directe du langage d'ontologie MVP-OWL selon la théorie des modèles

Dans cette section, nous présentons la sémantique par la théorie des modèles qui donne les interprétations sur la syntaxe abstraite du langage d'ontologie multi-points de vue `MVP-OWL` présentée dans la section *5.3.1*. La sémantique présentée ici est inspirée du travail de W3C « Direct Model-Theoretic Semantics »[1]. Les noms des vocabulaires, leurs interprétations et celles des constructeurs, des axiomes, des faits sont repris. Par contre, avec la présence de la notion de point de vue, de nouveaux vocabulaires, constructeurs sont introduits et les interprétations de `MVP-OWL` sont différentes de celles de `OWL`.

5.3.2.1 Vocabulaires et leurs interprétations

Comme `OWL`, une ontologie en `MVP-OWL` correspond à un vocabulaire. Ce vocabulaire comprend non seulement toutes les références URI et tous les littéraux de l'ontologie, mais aussi ceux des ontologies importées par l'ontologie en question et les autres références URI et autres littéraux.

> ***Définition 56*** *(Vocabulaire de MVP-OWL) Un vocabulaire de MVP-OWL, V, se compose d'un ensemble de littéraux V_L et de huit ensembles de références URI, V_C, V_D, V_I, V_{DP}, V_{IP}, V_{AP}, V_O et V_{VP}. V_{VP} est divisé en deux vocabulaires disjoints V_{VPP} et V_{VPO}. Dans tout vocabulaire V, V_C, V_D et V_{VP} sont deux à deux disjoints, V_{VPP} et V_{VPO} sont disjoints et V_{DP}, V_{IP}, V_{AP}, et V_{OP} sont deux à deux disjoints :*

[1] http://www.w3.org/TR/2004/REC-owl-semantics-20040210/direct.html

- *V_C est l'ensemble de noms de classe, et il contient owl:Thing et owl:Nothing.*
- *V_D est l'ensemble de noms de type de données, et il contient les références URI pour les types de données de OWL et rdfs:Literal.*
- *V_{AP} est l'ensemble de noms de propriété d'annotation, et il contient owl:versionInfo, rdfs:label, rdfs:comment, rdfs:seeAlso, et rdfs: isDefinedBy.*
- *V_{IP} est l'ensemble de noms de propriété ayant un individu comme valeur.*
- *V_{DP} est l'ensemble de noms de propriété ayant une donnée comme valeur.*
- *V_{OP} est l'ensemble de noms de propriété d'ontologie de OWL.*
- *V_I est l'ensemble de noms d'individu (instance).*
- *V_O est l'ensemble de noms d'ontologie, qui n'a pas besoin d'avoir un membre.*
- *V_{VP} est l'ensemble de noms des points de vue. Il est partitionné en deux ensembles disjoints : V_{VPP} l'ensemble de noms des points de vue perspectives, et V_{VPO} l'ensemble de noms des points de vue opinions. V_{VPP} contient le point de vue général VP_G (vp:GeneralViewpoint). $V_{VP}=V_{VPP} \cup V_{VPO}$, $V_{VPP} \cap V_{VPO} = \varnothing$.*

Les définitions des types de données (datatype) et des cartes de types de données (datatype map) sont similaires à celles de `OWL`.

Définition 57 *(Type de donnée) Un type de données d est caractérisé par (1) un espace lexical, L(d), qui est un ensemble de chaînes des caractères Unicode; (2) un espace de valeurs, V(d); et (3) une mise en correspondance totale (total mapping) L2V(d) de l'espace lexicologique vers l'espace de valeurs.*

Définition 58 *(Carte de types de données) Une carte de type de données D est une correspondance partielle (partial mapping) des références URIs avec les types de données qui met en correspondance xsd:string et xsd:integer avec les types de données appropriés de XML schéma.*

La définition de l'interprétation de la syntaxe abstraite de `MVP-OWL` en prenant en compte la notion de point de vue est donnée dans la *Définition 59*.

Définition 59 *(Interprétation de la syntaxe abstraite de MVP-OWL) Soit D une carte de types de données. Une interprétation de la syntaxe abstraite de MVP-OWL par rapport à D avec les vocabulaires V_L, V_C, V_D, V_I, V_{DP}, V_{IP}, V_{AP}, V_O et V_{VP} est un 7-tuple de la forme: $I = < R, EC, ER, L, S, LV, VP >$ où ($\mathcal{P}$ étant l'ensemble des partiels (power set operator)) :*

- *R (ensemble des ressources de I), est un ensemble non vide.*
- *O (ensemble des objets de I), est un ensemble non vide, inclus dans R et disjoint de LV. $O \subseteq R$, $O \cap LV = \varnothing$.*
- *LV (ensemble des valeurs littérales de I), est un sous-ensemble de R qui contient (1) l'ensemble de chaînes de caractères Unicode ; (2) l'ensemble de paires de chaînes de caractères Unicode et des étiquettes de langue ; et (3) les espaces de valeurs pour chaque type de données dans D.*

- *VP (ensemble des points de vue de I), est un sous-ensemble de R. VP est disjoint de O et de LV. $VP \subset R$. $VP \cap LV = \emptyset$, $VP \cap O = \emptyset$.*
- *VPP (ensemble des points de vue perspectives de I), est un sous-ensemble de VP. $VPP \subseteq VP$.*
- *VPO (ensemble des points de vue opinions de I), est un sous-ensemble de VP. VPO est disjoint de VPP. $VPO \subseteq VP$, $VPP \cap VPO = \emptyset$.*

- *EC : $V_C \rightarrow \mathcal{P}(O)$*
- *EC : $V_D \rightarrow \mathcal{P}(LV)$*
- *EC : $V_{VP} \rightarrow VP$*
- *EC : $V_{VPP} \rightarrow VPP$*
- *EC : $V_{VPO} \rightarrow VPO$*
- *EC : $V_{VP} \times V_{IP} \rightarrow \mathcal{P}(\mathcal{P}(O))$*
- *EC : $V_{VP} \times V_{DP} \rightarrow \mathcal{P}(\mathcal{P}(LV))$*
- *EC : $V_{VP} \times V_{AP} \rightarrow \mathcal{P}(\mathcal{P}(R))$*

- *ER : $V_{IP} \rightarrow \mathcal{P}(O \times O)$*
- *ER : $V_{DP} \rightarrow \mathcal{P}(O \times LV)$*
- *ER : $V_{AP} \cup \{rdf:type\} \rightarrow \mathcal{P}(R \times R)$*
- *ER : $V_{OP} \rightarrow \mathcal{P}(R \times R)$*
- *ER : $\{vp:characterisedBy\} \rightarrow VP \times R$*
- *ER : $\{vp:EQUIV, vp:INCL, vp:EXCL\} \rightarrow R \times R$*
- *ERVP : $V_{VP} \times V_{IP} \rightarrow \mathcal{P}(O \times O)$*
- *ERVP : $V_{VP} \times V_{DP} \rightarrow \mathcal{P}(O \times LV)$*
- *ERVP : $V_{VP} \times (V_{AP} \cup \{rdf:type\}) \rightarrow \mathcal{P}(R \times R)$*

- *L : TL $\rightarrow$ LV, où TL est l'ensemble de littéraux typés dans V_L*
- *S : $V_I \cup V_C \cup V_D \cup V_{DP} \cup V_{IP} \cup V_{AP} \cup V_O \cup V_{VP} \cup$ { owl:Ontology, owl:DeprecatedClass, owl:DeprecatedProperty } $\rightarrow R$*

- *$S(V_I) \subseteq O$*
- *EC(owl:Thing) = O*
- *EC(owl:Nothing) = { }*
- *EC(rdfs:Literal) = LV*
- *si D(d') = d alors EC(d') = V(d)*
- *si D(d') = d alors L("v"^^d') $\in$ V(d)*
- *si D(d') = d et v $\in$ L(d) alors L("v"^^d') = L2V(d)(v)*
- *si D(d') = d et v $\notin$ L(d) alors L("v"^^d') $\in R - LV$*

- *$\forall x \in O, \forall c \in V_C, x \in EC(c) \Leftrightarrow <x,S(c)> \in ER(rdf:type)$*
- *$S(vp:GeneralViewpoint) = VP_G \in VPP$*
- *$\forall vp \in V_{VP}, S(vp) \in VP$*
- *$S(V_{VP}) \subseteq VP$*
- *$vp \in V_{VP}, p \in V_{IP}, EC(vp, p) = \{a_i \mid \exists I \subset \mathbb{N}, i \in I, a_i \in \mathcal{P}(O)\}$. $\forall i \in I$, $a_i \subseteq EC(range(p))$*

- $vp \in V_{VP}$, $p \in V_{DP}$, $EC(vp, p) = \{a_i \mid \exists I \subset \mathbb{N}, i \in I, a_i \in \mathcal{P}(LV)\}$. $\forall i \in I, a_i \subseteq EC(range(p))$
- $vp \in V_{VP}$, $p \in V_{AP}$, $EC(vp, p) = \{a_i \mid \exists I \subset \mathbb{N}, i \in I, a_i \in \mathcal{P}(R)\}$. $\forall i \in I, a_i \subseteq EC(range(p))$
- $\forall p \in V_{IP} \cup V_{DP} \cup V_{AP}$, $\forall vpp \in V_{VPP}$, $ERVP(vpp, p) = ER(p)$
- $\forall p \in V_{IP} \cup V_{DP} \cup V_{AP}$, $ERVP(vp{:}GeneralViewpoint, p) = ER(p)$

EC fournit la signification pour les références URIs qui sont utilisées comme classes `MVP-OWL`, comme types de données et comme points de vue. EC est étendu pour fournir la signification pour des paires <vp, p> d'une référence URI utilisant comme un point de vue (vp) et une référence URI utilisant comme une propriété (p). La propriété dans une telle paire est soit une propriété ayant un individu comme valeur (V_{IP}), soit une propriété ayant une donnée comme valeur (V_{DP}), soit une propriété d'annotation (V_{AP}). L'interprétation pour chaque paire <vp, p> correspond à un ensemble des ensembles a_i des éléments qui sont des éléments dans l'espace des valeurs du co-domaine de la propriété p. Les ensembles a_i sont les sous-ensembles de l'espace des valeurs du co-domaine de la propriété p. Un ensemble a_i correspond à une façon de diviser l'espace des valeurs du co-domaine de la propriété p en un groupe des valeurs. Pour une même propriété p, des points de vue différents résultent de différentes interprétations correspondant aux différents ensembles ou aux différentes façons de diviser l'espace des valeurs du co-domaine de la propriété p.

Par exemple, examinons la propriété `hasAge` et deux points de vue vp_{Khaled} et vp_{Fabien}. Le co-domaine de la propriété `hasAge` est un nombre entier positif, supposons `range(hasAge)` = [0..200]. Selon le point de vue vp_{Khaled} ce co-domaine est divisé en deux groupes [0..17] et [18..200]. D'après le point de vue vp_{Fabien} ce co-domaine est divisé en trois groupes [0..18], [19..200] et [60..200]. Alors, les interprétations de la paire <vp, p> avec p est la propriété `hasAge` pour deux points de vue sont :

$$EC(<vp_{Khaled}, hasAge>) = \{ \{0, 1, \ldots, 17\}, \{18, 19, \ldots, 200\} \} = \{a_1, a_2\}$$
$$EC(<vp_{Fabien}, hasAge>) = \{ \{0, 1, \ldots, 18\}, \{19, 20, \ldots, 200\}, \{60, 61, \ldots, 200\} \} = \{b_1, b_2, b_3\}$$

Si le point de vue est un point de vue perspective, les interprétations des paires composées de ce point de vue et de n'importe quelle propriété sont cohérentes et non contradictoires au niveau global du modèle. Cela correspond aux interprétations des mêmes propriétés sans point de vue : $\forall vpp \in V_{VPP}$, $\forall p \in V_{IP} \cup V_{DP} \cup V_{AP}$, ERVP(vpp, p) = ER(p).

ER fournit la signification pour les références URIs qui sont utilisées comme propriétés `MVP-OWL` : les propriétés ayant un individu comme valeur (V_{IP}), les propriétés ayant une donnée comme valeur (V_{DP}), les propriétés d'annotation (V_{AP}) et les propriétés d'ontologies (V_{OP}).

ER est étendu pour fournir les significations des primitives de `MVP-OWL` : `vp:characterisedBy`, `vp:EQUIV`, `vp:INCL`, `vp:EXCL`. Comme nous l'avons présenté dans la section *5.2*, sur le modèle multi-points de vue MVP, `vp:EQUIV`, `vp:INCL`, `vp:EXCL` sont les propriétés de `MVP-OWL` qui relient deux classes ou propriétés, définies selon deux points de vue différents et expriment le lien entre ces deux entités : le lien d'équivalence, le lien d'inclusion et le lien d'exclusion, respectivement. La propriété

vp:EQUIV peut être utilisée pour relier deux instances équivalentes définies selon deux points de vue différents.

La propriété vp:characterisedBy permet de caractériser un point de vue par un critère qui est soit une propriété ayant un individu comme valeur (V_{IP}), soit une propriété ayant une donnée comme valeur (V_{DP}), soit une propriété d'annotation (V_{AP}). Dans un point de vue, on ne considère que les propriétés (critères) qui sont utilisées pour caractériser ce point de vue.

L fournit la signification pour les littéraux typés.

S fournit la signification pour les références URIs qui sont utilisées pour dénoter les individus de MVP-OWL, et permet de fournir la signification pour les annotations. S est étendu pour :

- Fournir la signification des références URIs qui sont utilisées pour référer les points de vue.
- Les littéraux pleins dans VL en les projetant sur eux-mêmes, c'est-à-dire, S("l") = l si l est un littéral plein sans étiquette de langue et S("l"@t) = < l,t > si l est un littéral plein avec une étiquette de langue.
- Les littéraux typés en employant L, S(l) = L(l) si l est un littéral typé.

ERVP fournit la signification pour les références URIs qui sont utilisées comme propriétés MVP-OWL *dans un point de vue*. Une relation particulière (représentée par une propriété) peut être établie entre deux individus réels dans un point de vue mais pas dans un autre point de vue. Par exemple, si pour le point de vue de Khaled (vp_{Khaled}), la voiture de Rose (Voiture_Rose) a une couleur bleue (Couleur_Bleue) et si d'après le point de vue de Fabien (vp_{Fabien}), la même voiture de Rose (Voiture_Rose) a une couleur rouge (Couleur_Rouge), alors <S(Voiture_Rose), S(Couleur_Bleue)> ∈ ERVP (vp_{Khaled}, aCouleur) et <S(Voiture_Rose), S(Couleur_Rouge)> ∈ ERVP(vp_{Fabien}, aCouleur), mais <S(Voiture_Rose), S(Couleur_Bleue)> ∉ ERVP(vp_{Fabien}, aCouleur). Autrement dit, la paire <S(Voiture_Rose), S(Couleur_Bleue)>, représentant la relation « avoir une couleur » (aCouleur) entre deux individus Voiture_Rose et Couleur_Bleue, n'appartient qu'à l'interprétation de la propriété aCouleur dans le point de vue de Khaled vp_{Khaled}, et pas dans le point de vue de Fabien ; et vice-versa la paire <S(Voiture_Rose), S(Couleur_Rouge)> n'appartient qu'à l'interprétation de aCouleur dans le point de vue de Fabien vp_{Fabien} et pas de Khaled vp_{Khaled}. Cela permet d'exprimer des valeurs différentes pour un même attribut (propriété) sur un même objet réel selon différents points de vue.

La propriété rdf:type est ajoutée dans l'ensemble des propriétés d'annotation afin de fournir une signification pour le type d'une instance selon un point de vue. ERVP(vp, rdf:type) est une interprétation de la propriété rdf:type selon le point de vue vp, qui est un ensemble des paires <S(i), S(c)> où S(i) est une instance, S(c) est une classe, et i instancie (est du type de) c *selon* le point de vue vp. Pour les points de vue différents vp_i, les interprétations (les ensembles des paires <S(i), S(c)>) de ERVP(vp_i, rdf:type) peuvent être différentes.

Les points de vue perspectives (V_{VPP}), incluent le point de vue général VP_G (représenté par `vp:GeneralViewpoint`), qui permet d'exprimer des valeurs consensuelles des propriétés à propos d'un objet réel. Contrairement aux points de vue opinions (V_{VPO}) des personnes ou des groupes, l'interprétation des propriétés selon les points de vue perspectives est pertinente et valide pour tous. Par exemple, si tous affirment que Fabien est un homme, nous représentons : <S(`Fabien`), S(`Homme`)> ∈ ERVP(`vp:GeneralViewpoint`, `rdf:type`) = ER(`rdf:type`).

5.3.2.2 Interprétation des constructeurs

EC est étendu aux constructeurs syntaxiques des descriptions, des portées des données, des individus, des valeurs, et des annotations comme dans le *Tableau 20*. Dans ce tableau, nous utilisons `c`, `p`, `i` pour dénoter une classe, une propriété et une instance, respectivement ; `vp`, `vpp`, `vpo` pour dénoter un point de vue (qui peut être un point de vue perspective ou un point de vue opinion), un point de vue perspective et un point de vue opinion, respectivement.

Syntaxe abstraite	Interprétation (valeur de EC)
classWithViewpoint(vpp c) pour vpp un point de vue perspective	EC(*classWithViewpoint*(vpp c p_1)) ∩ … ∩ EC(*classWithViewpoint*(vpp c p_n)) <vpp,p_i> ∈ ER(*vp:characterisedBy*) pour $1 \leq i \leq n$
classWithViewpoint(vpp c p) pour p ∈ $V_{IP} \cup V_{DP} \cup V_{AP}$	{x ∈ O | x ∈ EC(c) ∧ ∃i <x,y> ∈ ER(p), y ∈ a_i, a_i ∈ EC(vpp,p)}
complementOf(c)	O – EC(c)
unionOf(c_1 … c_n)	EC(c_1) ∪ … ∪ EC(c_n)
intersectionOf(c_1 … c_n)	EC(c_1) ∩ … ∩ EC(c_n)
oneOf(i_1 … i_n), pour i_j IDs individus	{S(i_1), …, S(i_n)}
oneOf(v_1 … v_n), pour v_j littéraux	{S(v_1), …, S(v_n)}
restriction(p x_1 … x_n), pour n > 1	ECVP(*restriction*(p x_1)) ∩ … ∩ ECVP(*restriction*(p x_n))
restriction(p *allValuesFrom*(r))	{x ∈ O | <x,y> ∈ ER(p) ⇒ y ∈ EC(r)}
restriction(p *someValuesFrom*(e))	{x ∈ O | ∃ <x,y> ∈ ER(p) ∧ y ∈ EC(e)}
restriction(p *value*(i)), pour i un ID individu	{x ∈ O | <x,S(i)> ∈ ER(p)}
restriction(p *value*(v)), pour v un litéral	{x ∈ O | <x,S(v)> ∈ ER(p)}
restriction(p *minCardinality*(n))	{x ∈ O | card({y ∈ O∪LV : <x,y> ∈ ER(p)}) ≥ n}
restriction(p *maxCardinality*(n))	{x ∈ O | card({y ∈ O∪LV : <x,y> ∈ ER(p)}) ≤ n}
restriction(p *cardinality*(n))	{x ∈ O | card({y ∈ O∪LV : <x,y> ∈ ER(p)}) = n}
Individual(vp *annotation*(q_1 o_1) … *annotation*(q_k o_k) *type*(c_1) … *type*(c_m) (p_1 v_1) … (p_n v_n))	{x ∈ R | <x,S(o_1)> ∈ ERVP(vp,q_1)} ∩ … ∩ {x ∈ R | <x,S(o_k)> ∈ ERVP(vp,q_k)} {x ∈ O | <x,S(c_1)> ∈ ERVP(vp,*rdf:type*)} ∩ … ∩ {x ∈ O | <x,S(c_m)> ∈ ERVP(vp,*rdf:type*)} ∩ EC(*value*(vp p_1 v_1)) ∩ … ∩ EC(*value*(vp p_n v_n))
Individual(i vp *annotation*(q_1 o_1) … *annotation*(q_k o_k) *type*(c_1) … *type*(c_m) (p_1 v_1) … (p_n v_n))	{S(i)} ∩ {x ∈ R | <x,S(o_1)> ∈ ERVP(vp,q_1)} ∩ … ∩ {x ∈ R | <x,S(o_k)> ∈ ERVP(vp,q_k)} {x ∈ O | <x,S(c_1)> ∈ ERVP(vp,*rdf:type*)} ∩ … ∩ {x ∈ O | <x,S(c_m)> ∈ ERVP(vp,*rdf:type*)} ∩ EC(*value*(vp p_1 v_1)) ∩ … ∩ EC(*value*(vp p_n v_n))
Individual([i] *annotation*(q_1 o_1) … *annotation*(q_k o_k)	EC(*Individual*([i] *vp:GeneralViewpoint* *annotation*(q_1 o_1) … *annotation*(q_k o_k)

type(c_1) … *type*(c_m) (p_1 v_1) … (p_n v_n))	*type*(c_1) … *type*(c_m) (p_1 v_1) … (p_n v_n))) = [{S(i)} ∩] {x ∈ R | <x,S(o_1)> ∈ ER(q_1)} ∩ … ∩ {x ∈ R | <x,S(o_k)> ∈ ER(q_k)} ∩ EC(c_1) ∩ … ∩ EC(c_m) ∩ EC(*value*(p_1 v_1)) ∩ … ∩ EC(*value*(p_n v_n))
value(vp p *Individual*(…))	{x ∈ O | ∃ y ∈ EC(*Individual*(…)) : <x,y> ∈ ERVP(vp,p)}
value(p *Individual*(…))	EC(*value*(*vp:GeneralViewpoint* p *Individual*(…))) = {x ∈ O | ∃ y ∈ EC(*Individual*(…)): <x,y> ∈ ER(p)}
value(vp p id) pour id un ID individu	{x ∈ O | <x,S(id)> ∈ ERVP(vp,p)}
value(p id) pour id un ID individu	EC(*value*(*vp:GeneralViewpoint* p id)) = {x ∈ O | <x,S(id)> ∈ ER(p)}
value(vp p v) pour v un litéral	{x ∈ O | <x,S(v)> ∈ ERVP(vp,p)}
value(p v) pour v un litéral	EC(*value*(*vp:GeneralViewpoint* p v)) = {x ∈ O | <x,S(v)> ∈ ER(p)}
annotation(p o) pour o une reference URI	{x ∈ R | <x,S(o)> ∈ ER(p)}
annotation(p *Individual*(…))	{x ∈ R | ∃ y ∈ EC(*Individual*(…)): <x,y> ∈ ER(p)}

Tableau 20 Extension de EC

5.3.2.3 Interprétation des axiomes et des faits

Une interprétation abstraite de MVP-OWL, I, satisfait des axiomes et des faits de MVP-OWL comme cela est indiqué dans le *Tableau 21*.

Directive	Conditions sur des interprétations
EQUIV(c_1 c_2) pour c_1 c_2 deux classes	S(c_1) = S(c_2) ∀ x ∈ O, x ∈ EC(c_1) ⇔ x ∈ EC(c_2) EC(c_1) = EC(c_2) ER(*owl:equivalentClass*) ⊆ ER(*vp:EQUIV*)
EQUIV(p_1 p_2) pour p_1 p_2 deux propriétés	S(p_1) = S(p_2) ∀ <x,y> ∈ R×R, <x,y> ∈ ER(p_1) ⇔ <x,y> ∈ ER(p_2) ER(p_1) = ER(p_2) ER(*owl:equivalentProperty*) ⊆ ER(*vp:EQUIV*)
EQUIV(i_1 i_2) pour i_1 i_2 deux instances	S(i_1) = S(i_2) ER(*owl:sameAs*) ⊆ ER(*vp:EQUIV*)
INCL(c_1 c_2) pour c_1 c_2 deux classes	∀ x ∈ O, x ∈ EC(c_2) ⇒ x ∈ EC(c_1) EC(c_2) ⊆ EC(c_1) ∀ <a,b> ∈ O×O, <a,b> ∈ ER(*rdfs:subClassOf*) ⇒ <b,a> ∈ ER(*vp:INCL*)
INCL(p_1 p_2) pour p_1 p_2 deux propriétés	∀ <x,y> ∈ O×O, <x,y> ∈ ER(p_2) ⇒ <x,y> ∈ ER(p_1) ER(p_2) ⊆ ER(p_1) ∀ <a,b> ∈ O×O, <a,b> ∈ ER(*rdfs:subPropertyOf*) ⇒ <b,a> ∈ ER(*vp:INCL*)
EXCL(c_1 c_2) pour c_1 c_2 deux classes	S(c_1) ≠ S(c_2) ∀ x ∈ O, x ∈ EC(c_1) ⇒ x ∉ EC(c_2) ∀ x ∈ O, x ∈ EC(c_2) ⇒ x ∉ EC(c_1)

	$EC(c_1) \cap EC(c_2) = \emptyset$ $\forall$ <a,b> $\in$ O×O, <a,b> $\in$ ER(*owl:disjointWith*) $\Rightarrow$ <a,b> $\in$ ER(*vp:EXCL*) $\wedge$ <b,a> $\in$ ER(*vp:EXCL*) $\exists$ c $\in V_C$, $\exists$ vpp $\in V_{VPP}$, $EC(c_1) \subseteq EC($*classWithViewpoint*(vpp c)) $EC(c_2) \subseteq EC($*classWithViewpoint*(vpp c))
Viewpoint(vp *annotation*(q_1 o_1) … *annotation*(q_k o_k) *characterisedBy*(p_1) … *characterisedBy*(p_n))	$S(vp) \in VP$ $S(vp) \in EC($*annotation*(q_1 o_1)) … $S(vp) \in EC($*annotation*(q_k o_k)) <$S(vp),S(p_1)$> $\in EC($*vp:characterisedBy*) … <$S(vp),S(p_n)$> $\in EC($*vp:characterisedBy*) $\forall i$ $ER(p_i) \subseteq O \times LV \cup O \times O \cup R \times R$
Class(c [*Deprecated*] *complete* *annotation*(p_1 o_1) … *annotation*(p_k o_k) $descr_1$ … $descr_n$)	[<S(c),S(*owl:DeprecatedClass*)> $\in$ ER(*rdf:type*)] $S(c) \in EC($*annotation*(p_1 o_1)) … $S(c) \in EC($*annotation*(p_k o_k)) $EC(c) = EC(descr_1) \cap \ldots \cap EC(descr_n)$
Class(c [*Deprecated*] *partial* *annotation*(p_1 o_1) … *annotation*(p_k o_k) $descr_1$ … $descr_n$)	[<S(c),S(*owl:DeprecatedClass*)> $\in$ ER(*rdf:type*)] $S(c) \in EC($*annotation*(p_1 o_1)) … $S(c) \in EC($*annotation*(p_k o_k)) $EC(c) \subseteq EC(descr_1) \cap \ldots \cap EC(descr_n)$
EnumeratedClass(c [*Deprecated*] *annotation*(p_1 o_1) … *annotation*(p_k o_k) i_1 … i_n)	[<S(c),S(*owl:DeprecatedClass*)> $\in$ ER(*rdf:type*)] $S(c) \in EC($*annotation*(p_1 o_1)) … $S(c) \in EC($*annotation*(p_k o_k)) $EC(c) = \{S(i_1), \ldots, S(i_n)\}$
Datatype(c [*Deprecated*] *annotation*(p_1 o_1) … *annotation*(p_k o_k))	[<S(c),S(*owl:DeprecatedClass*)> $\in$ ER(*rdf:type*)] $S(c) \in EC($*annotation*(p_1 o_1)) … $S(c) \in EC($*annotation*(p_k o_k)) $EC(c) \subseteq LV$
DisjointClasses(d_1 … d_k) pour k > 1	$EC(d_i) \cap EC(d_j) = \{\}$ pour $1 \le i < j \le n$
EquivalentClasses(d_1 … d_k) pour k > 1	$EC(d_i) = EC(d_j)$ pour $1 \le i < j \le n$
SubClassOf(d_1 d_2)	$EC(d_1) \subseteq EC(d_2)$
DatatypeProperty(p [*Deprecated*] *annotation*(p_1 o_1) … *annotation*(p_k o_k) *super*(s_1) … *super*(s_n) *domain*(d_1) … *domain*(d_n) *range*(r_1) … *range*(r_n) [*Functional*])	[<S(p),S(*owl:DeprecatedProperty*)> $\in$ ER(*rdf:type*)] $S(p) \in EC($*annotation*(p_1 o_1)) … $S(p) \in EC($*annotation*(p_k o_k)) $ER(p) \subseteq O \times LV \cap ER(s_1) \cap \ldots \cap ER(s_n) \cap$ $EC(d_1) \times LV \cap \ldots \cap EC(d_n) \times LV \cap$ $O \times EC(r_1) \cap \ldots \cap O \times EC(r_n)$ [ER(p) is functional]
ObjectProperty(p [*Deprecated*] *annotation*(p_1 o_1) … *annotation*(p_k o_k) *super*(s_1) … *super*(s_n) *domain*(d_1) … *domain*(d_n) *range*(r_1) … *range*(r_n) [*inverse*(i)] [*Symmetric*] [*Functional*] [*InverseFunctional*] [*Transitive*])	[<S(p),S(*owl:DeprecatedProperty*)> $\in$ ER(*rdf:type*)] $S(p) \in EC($*annotation*(p_1 o_1)) … $S(p) \in EC($*annotation*(p_k o_k)) $ER(p) \subseteq O \times O \cap ER(s_1) \cap \ldots \cap ER(s_n) \cap$ $EC(d_1) \times O \cap \ldots \cap EC(d_n) \times O \cap$ $O \times EC(r_1) \cap \ldots \cap O \times EC(r_n)$ [ER(p) is the inverse of ER(i)] [ER(p) is symmetric] [ER(p) is functional] [ER(p) is inverse functional] [ER(p) is transitive]
AnnotationProperty(p *annotation*(p_1 o_1) … *annotation*(p_k o_k))	$S(p) \in EC($*annotation*(p_1 o_1)) … $S(p) \in EC($*annotation*(p_k o_k))
OntologyProperty(p *annotation*(p_1 o_1) … *annotation*(p_k o_k))	$S(p) \in EC($*annotation*(p_1 o_1)) … $S(p) \in EC($*annotation*(p_k o_k))
EquivalentProperties(p_1 … p_n) pour n > 1	$ER(p_i) = ER(p_j)$ pour $1 \le i < j \le n$
SubPropertyOf(p_1 p_2)	$ER(p_1) \subseteq ER(p_2)$
SameIndividual(i_1 … i_n) pour n > 1	$S(i_j) = S(i_k)$ pour $1 \le j < k \le n$
SameIndividual(vpp i_1 … i_n) pour n > 1	$S(i_j) = S(i_k)$ <i_j,i_k> $\in$ ER(*owl:sameAs*) pour $1 \le j < k \le n$
SameIndividual(vpo i_1 … i_n) pour n > 1	<i_j,i_k> $\in$ ERVP(vpo,*owl:sameAs*) pour $1 \le j < k \le n$
DifferentIndividuals(i_1 … i_n) pour n > 1	$S(i_j) \neq S(i_k)$ pour $1 \le j < k \le n$

DifferentIndividuals(vpp i_1 … i_n) pour n > 1	$S(i_j) \neq S(i_k)$ <i_j,i_k> ∈ ER(*owl:differentFrom*) pour 1 ≤ j < k ≤ n
DifferentIndividuals(vpo i_1 … i_n) pour n > 1	<i_j,i_k> ∈ ERVP(vpo,*owl:differentFrom*) pour 1 ≤ j < k ≤ n
Individual([i] *annotation*(p_1 o_1) … *annotation*(p_k o_k) *type*(c_1) … *type*(c_m) (p_1 v_1) … (p_n v_n))	EC(*Individual*([i] *annotation*(p_1 o_1) … *annotation*(p_k o_k) *type*(c_1) … *type*(c_m) (p_1 v_1) … (p_n v_n)) est non vide
Individual([i] vp *annotation*(p_1 o_1) … *annotation*(p_k o_k) *type*(c_1) … *type*(c_m) (p_1 v_1) … (p_n v_n))	EC(*Individual*([i] vp *annotation*(p_1 o_1) … *annotation*(p_k o_k) *type*(c_1) … *type*(c_m) (p_1 v_1) … (p_n v_n)) est non vide

Tableau 21 Interprétation des axiomes et des faits

5.3.2.4 Interprétation de l'ontologie en MVP-OWL

Définition 60 *(Interprétation de l'ontologie en MVP-OWL) Soit D une carte de types de données. Une interprétation abstraite de MVP-OWL, I, par rapport à D avec les vocabulaires V_L, V_C, V_D, V_I, V_{DP}, V_{IP}, V_{AP}, V_O et V_{VP}, satisfait une ontologie MVP-OWL, O, ssi :*

1. *Chaque référence URI dans O utilisée comme un ID (identificateur) de classe (type de donnée, individu, propriété ayant une donnée comme valeur, propriété ayant un individu comme valeur, propriété d'annotation, propriété d'ontologie, et point de vue) appartient à V_C (V_D, V_I, V_{DP}, V_{IP}, V_{AP}, V_O et V_{VP}, respectivement).*
2. *Chaque littéral dans O appartient à V_L.*
3. *I satisfait chaque directive dans O, excepté les annotations d'ontologie.*
4. *Il existe o ∈ R avec <o, S(owl:Ontology)> ∈ ER(rdf:type) tel que pour chaque annotation d'ontologie de la forme Annotation(p v), <o, S(v)> ∈ ER(p) et tel que si O a nom n, alors S(n) = o.*
5. *I satisfait chaque ontologie mentionnée dans une directive d'annotation owl:imports de O.*

5.4 Construction d'ontologies multi-points de vue

La construction d'une ontologie peut être effectuée de plusieurs façons : construction à partir de zéro, réutilisation et re-ingénierie (re-engineering) d'autres ontologies, fusion d'ontologies ou approche d'apprentissage d'ontologie. Différentes méthodes et méthodologies pour construire une ontologie sont présentées et analysées dans [Gómez-Pérez *et al.*, 2004]. Dans ce travail, les auteurs présentent les activités concernant la construction d'ontologie. Les activités sont classées en trois catégories : la gestion d'ontologies (les activités concernant le planning des tâches, le contrôle, et l'assurance de qualité) ; le développement d'ontologies ; et les activités supportant l'ontologie telles que l'acquisition des connaissances, l'évaluation, l'intégration, la fusion, l'alignement, la documentation et la gestion de configuration. La deuxième catégorie à son tour est divisée en trois groupes : le pré-développement (l'étude de l'environnement, l'étude de faisabilité), le développement (la spécification, la conceptualisation, la formalisation et l'implémentation), le post-développement (la maintenance et l'utilisation).

Comme pour ontologie classique, les méthodes et les méthodologies pour construire une ontologie multi-points de vue peuvent inclure toutes les activités intervenant dans le processus de développement d'ontologie ci-dessus. Cependant, la notion de point de vue vise à permettre d'exprimer différentes descriptions et explications des experts à propos des connaissances dans le domaine. Elle intervient donc lors de certaines activités dans le processus, telles que l'acquisition des connaissances, l'intégration ou la fusion des ontologies déjà existantes (dans la troisième catégorie) ; ou la spécification, la conceptualisation, la formalisation, l'implémentation de catégorie de développement. Par exemple, pour l'activité d'acquisition des connaissances, différentes opinions des experts du domaine sont prises en compte et analysées pour déterminer les différents points de vue possibles. La fusion de deux ontologies peut considérer des entités de deux ontologies qui sont définies différemment mais compatibles comme une seule entité dans l'ontologie fusionnée avec plusieurs descriptions selon plusieurs différents points de vue. De même, dans l'activité de conceptualisation, dont le but est de structurer les connaissances de domaine comme des modèles significatifs au niveau de connaissance [Newell, 1982], la notion de point de vue intervient dans la façon de structurer le modèle, dans les places des concepts, dans la taxonomie des concepts et dans les termes permettant de dénoter les concepts.

Néanmoins, dans cette section, nous n'envisageons pas de présenter une méthode ou une méthodologie de construction d'une ontologie multi-points de vue qui inclue toutes les activités dans le processus de développent d'ontologie. Nous étudions plutôt des *techniques* nécessaires pour permettre d'implémenter une ontologie multi-points de vue en langage `MVP-OWL`. Selon [IEEE, 1990], une technique est « *une procédure technique et gestionnaire employée pour atteindre un objectif donné* », tandis qu'une méthodologie est « *une série exhaustive et intégrée des techniques ou des méthodes créant une théorie de systèmes généraux de la façon dont une classe de travail pensée-intensif doit être exécutée* » et une méthode est un ensemble « *des processus ou des procédures ordonnés utilisés dans l'ingénierie d'un produit ou dans la réalisation d'un service* ».

Nous présentons dans les sections suivantes des techniques, des opérateurs permettant d'implémenter et d'exploiter une ontologie multi-points de vue en MVP-OWL, ainsi qu'une méthode de construction d'une ontologie multi-points de vue à partir d'ontologies déjà existantes en exploitant des résultats des travaux de recherche des correspondances entre des ontologies.

5.4.1 Opérateurs de MVP-OWL pour la construction d'ontologie multi-points de vue

Les opérateurs de base pour la construction d'ontologie incluent des opérateurs de création, de suppression, de modification des entités dans l'ontologie. Dans notre modèle MVP, outre des entités de l'ontologie normale (i.e. classe, propriété, instance), il y a aussi des entités « point de vue ».

L'ontologie multi-points de vue en MVP-OWL correspond à un ensemble de triplets RDF, noté O. Chaque description dans la définition d'une entité correspond à un triplet RDF. Les opérateurs pour la construction d'ontologie multi-points de vue modifient donc les triplets dans l'ensemble O.

Algorithme 23 Ajouter_Entité(O, entitéID, entitéType, {$desc_i$ vp_i})

```
Pour chaque i
  Si entitéType == owl:Class et
     desc_i.prédicat == rdfs:subClassOf et
     vp_i non nul
    s_vp <- créer une nouvelle ressource
    Ajouter_Triplet(O, <entitéID rdfs:subClassOf s_vp>)
    Ajouter_Triplet(O, <s_vp vp:onViewpoint vp_i>)
    Ajouter_Triplet(O, <s_vp vp:onClass desc_i.objet >)
  Sinon
    Ajouter_Triplet(O, <entitéID desc_i.prédicat desc_i.objet>)
  Fin Si
Fin Pour

Retourner
```

L'*Algorithme 23* permet d'ajouter une entité (classe, propriété, instance et point de vue) dans l'ontologie. L'entrée est un ensemble de triplets O correspondant à l'ontologie, l'identificateur de l'entité (URI) entitéID, son type entitéType (owl:Class pour une classe, owl:ObjectProperty pour une propriété, une URI d'une classe pour définir une instance de cette classe et vp:Viewpoint pour un point de vue) et un ensemble de paires $desc_i$ = <$prédicat_i$ $objet_i$> qui sont les descriptions décrivant l'entité. Le prédicat $prédicat_i$ est une des primitives de MVP-OWL et l'objet $objet_i$ est sa valeur. $desc_i$.prédicat et $desc_i$.objet permettent d'accéder au prédicat et à l'objet de la paire $desc_i$, respectivement. L'algorithme examine toutes les $desc_i$.

La description de sous-classe d'une classe `C` selon un point de vue est effectuée en spécifiant la $desc_i$ = <`rdfs:subClassOf C`> et vp_i est le point de vue en question. Dans ce cas, l'algorithme crée les triplets appropriés représentant la défintion d'une classe avec un point de vue. Dans le cas contraire, l'algorithme ajoute simplement le triplet <`entitéID` $desc_i$`.prédicat` $desc_i$`.objet`> dans l'ensemble de triplets `O` représentant l'ontologie.

L'*Algorithme 24* permet de supprimer des descriptions à propos d'une entité. L'entrée est un ensemble de triplets `O` correspondant à l'ontologie, l'identificateur de l'entité (URI) `entitéID` et un ensemble de paires $desc_i$ = <$prédicat_i$ $objet_i$> qui sont les descriptions décrivant l'entité à supprimer.

Si l'entité à supprimer est une classe et la description à supprimer est une description de sous-classe d'une classe selon un point de vue, l'algorithme cherche la ressource anonyme représentant la définition de la classe mère ($desc_i$`.objet`) avec le point de vue (vp_i), ensuite il supprime les triplets appropriés. Dans le cas contraire, l'algorithme enlève simplement le triplet <`entitéID` $desc_i$`.prédicat` $desc_i$`.objet`> de l'ensemble de triplets `O` représentant l'ontologie.

Algorithme 24 Supprimer_Entité(O, entitéID, { $desc_i$ vp_i })

```
Pour chaque i
  Si Type(entitéID) == owl:Class et
     descᵢ.prédicat == rdfs:subClassOf et
     vpᵢ non nul
    s_vp <- ChercherRessourceAnonyme(descᵢ.objet, vpᵢ)
    Si s_vp existe
      Supprimer_Tripler(O, <entitéID rdfs:subClassOf s_vp>)
      Supprimer_Triplet(O, <s_vp vp:onViewpoint vpᵢ>)
      Supprimer_Tripler(O, <s_vp vp:onClass descᵢ.objet>)
    Fin Si
  Sinon
    Supprimer_Tripler(O, <entitéID descᵢ.prédicat descᵢ.objet>)
  Fin Si
Fin Pour

Retourner
```

La modification des descriptions d'une entité est en fait une combinaison de la suppression et l'ajout des descriptions correspondantes. Rappelons que le point de vue, comme la classe, la propriété, l'instance, est une entité du modèle `MVP`. Les algorithmes *Algorithme 23* et *Algorithme 24* permettent donc non seulement de créer, de supprimer, de modifier les descriptions des classes, des propriétés, des instances, mais aussi les descriptions des points de vue.

5.4.2 Méthode pour la construction d'ontologie multi-points de vue à partir des ontologies déjà existantes

Les méthodes pour construire une nouvelle ontologie à partir des ontologies sont étudiées dans ONIONS [Steve *et al.*, 1997 ; Gangemi *et al.*, 1999], FCA-Merge [Stumme et Maedche, 2001], PROMPT [Noy et Musen, 2000]. Ce sont des méthodes de fusion des ontologies. La méthode ONIONS permet de créer une librairie d'ontologies à partir de différentes sources et de les connecter par des termes des théories génériques partagées. ONIONS se compose de 10 activités pour effectuer *manuellement* la fusion des ontologies. Quant à FCA-Merge, la fusion de deux ontologies se base sur un ensemble de documents dans les domaines représentés par les ontologies. La fusion est réalisée en extrayant des instances à partir des documents qui appartiennent aux concepts de deux ontologies. Deux concepts de deux ontologies sont considérés comme un même concept dans l'ontologie fusionnée s'ils ont des mêmes instances dans les mêmes documents. Enfin, la méthode PROMPT est une méthode de fusion semi-automatique (avec interaction humaine). Elle se compose de 5 étapes qui sont effectuées dans un processus cyclique : (1) le système crée une liste des opérations de fusion suggérées à partir de deux ontologies ; l'ontologiste (2) sélectionne une opération et (3) l'exécute ; puis, (4) le système génère une liste des conflits résultant de l'exécution de l'opération ; (5) le système met à jour la liste des opérations à exécuter dans l'itération suivante selon la résolution de conflits choisie.

En s'inspirant des approches présentées ci-dessus, avec l'objectif de construire une nouvelle ontologie multi-points de vue fusionnée à partir de deux ontologies sans points de vue déjà existantes, nous proposons notre méthode qui se compose des étapes suivantes :

- Chercher des entités correspondantes (qui représentent un même concept ou un même objet) entre deux ontologies en employant des algorithmes d'alignement d'ontologies tels que ASCO1, ASCO2 ou ASCO3. Le résultat est une liste L des paires des entités similaires.

- Créer les points de vue vp_1 et vp_2 dans l'ontologie multi-points de vue fusionnée O_M, qui représentent les points de vue correspondant à l'ontologie O_1 et O_2, respectivement.

- Pour chaque entité e_1, qui est soit une classe, soit une propriété, soit une instance, dans la première ontologie O_1 :

  - S'il existe une entité e_2, de même type que l'entité e_1, de la deuxième ontologie O_2, dont la paire $<e_1, e_2>$ appartient à la liste L (elles représentent un même concept ou un même objet), alors :

    - Créer l'entité e_M, de même type que l'entité e_1 et e_2, dans l'ontologie multi-points de vue fusionnée O_M

    - Si l'entité e_M est une instance

- Ajouter les descriptions dans la définition de e_1, à la définition de e_M, sous le point de vue vp_1
- Ajouter les descriptions dans la définition de e_2, à la définition de e_M, sous le point de vue vp_2

▪ Si l'entité e_M est une propriété :

- Ajouter les descriptions dans la définition de e_1, à la définition de e_M
- Ajouter les descriptions dans la définition de e_2, qui ne correspondent à aucune description dans la définition de e_1, à la définition de e_M

▪ Si l'entité e est une classe :

- Ajouter les descriptions, qui n'utilisent pas la primitive `rdfs:subClassOf`, dans la définition de e_1, à la définition de e_M
- Ajouter les descriptions, qui n'utilisent pas la primitive `rdfs:subClassOf`, dans la définition de e_2, à la définition de e_M, si ces descriptions n'existent pas encore dans la définition de e_M
- Pour les descriptions de subsomption, qui utilisent la primitive `rdfs:subClassOf`, dans la définition de e_1 et e_2, nous les ajoutons à la définition de e_M selon le point de vue vp_1 et vp_2, respectivement, en créant les ressources anonymes et en utilisant les primitives `vp:onViewpoint` et `vp:onClass`.

- Sinon :

▪ Créer l'entité e_M de même type que l'entité e_1, dans l'ontologie multi-points de vue fusionnée O_M

▪ Si e_1 est une instance, ajouter les descriptions dans la définition de e_1, à la définition de e_M, sous le point de vue vp_1

▪ Si e_1 est une propriété, ajouter les descriptions dans la définition de e_1, à la définition de e_M

▪ Si e_1 est une classe :

- Ajouter des descriptions de subsomption, qui utilisent la primitive `rdfs:subClassOf`, dans la définition de e_1, à la définition de e_M, sous le point de vue vp_1
- Pour d'autres descriptions dans la définition de e_1, les ajouter à la définition de e_M

- Pour toutes les entités e_{2j} de la deuxième ontologie O_2 qui n'ont pas encore été traitées dans l'étape précédente (les entités ne sont pas dans la liste L)

▪ Créer l'entité e_M de même type que l'entité e_2, dans l'ontologie multi-points de vue fusionnée O_M

▪ Si e_{2j} est une instance, ajouter les descriptions dans la définition de e_{2j}, à la définition de e_M, sous le point de vue vp_2

- Si e_{2j} est une propriété, ajouter les descriptions dans la définition de e_{2j}, à la définition de e_M
- Si e_{2j} est une classe :
  - Ajouter des descriptions de subsomption, qui utilisent la primitive `rdfs:subClassOf`, dans la définition de e_{2j}, à la définition de e_M, sous le point de vue vp_2
  - Pour d'autres descriptions dans la définition de e_{2j}, nous les ajoutons à la définition de e_M

La méthode proposée est donc une approche automatique permettant de construire une ontologie multi-points de vue à partir des ontologies déjà existantes. Le résultat, l'ontologie multi-points de vue, dépend de la qualité du résultat de la tâche d'alignement d'ontologie.

L'intégration automatique d'une ou plusieurs autres ontologies normales avec l'ontologie multi-points de vue requiert l'alignement automatique entre une ontologie multi-points de vue et une ontologie normale. Cette tâche est une tâche plus difficile de celle de l'alignement d'ontologies, présentée dans les chapitres 2 et 3. Cependant, elle peut être effectuée manuellement avec l'aide des ontologistes et des experts de domaine, selon la méthode suivante :

- Créer le point de vue vp_i dans l'ontologie multi-points de vue O_M, qui représente le point de vue correspondant à l'ontologie O_i.
- Pour chaque entité e, qui est soit une classe, soit une propriété, soit une instance, dans l'ontologie O_i à intégrer dans l'ontologie multi-points de vue O_M :
  - Détecter manuellement l'entité e_M dans l'ontologie O_M qui correspond à l'entité e (elles représentent un même concept ou un même objet). S'il existe une telle entité :
    - Si l'entité e est une instance, ajouter les descriptions dans la définition de e, à la définition de e_M, sous le point de vue vp_i
    - Si l'entité e est une propriété :
      - Chercher des descriptions d_k dans la définition de e qui ne correspondent à aucune description dans la définition de e_M
      - Ajouter ces descriptions d_k à la définition de e_M
    - Si l'entité e est une classe :
      - Ajouter les descriptions, qui n'utilisent pas la primitive `rdfs:subClassOf`, dans la définition de e, à la définition de e_M, si ces descriptions n'existent pas encore dans la définition de e_M
      - Pour les descriptions de subsomption, qui utilisent la primitive `rdfs:subClassOf`, dans la définition de e, les ajouter à la définition de e_M selon le point de vue vp_i en créant les ressources anonymes et en utilisant les primitives `vp:onViewpoint` et `vp:on-Class`.

- Sinon :
  - Créer l'entité e_M de même type que l'entité e, dans l'ontologie multi-points de vue O_M
  - Si l'entité e est une instance, ajouter les descriptions dans la définition de e, à la définition de e_M, sous le point de vue vp_i
  - Si l'entité e est une propriété, ajouter les descriptions dans la définition de e, à la définition de e_M
  - Si l'entité e est une classe :
    - Ajouter des descriptions, qui n'utilisent pas la primitive `rdfs:subClassOf`, dans la définition de e, à la définition de e_M
    - Pour les descriptions de subsomption, qui utilisent la primitive `rdfs:subClassOf`, dans la définition de e, les ajouter à la définition de e_M selon le point de vue vp_i en créant les ressources anonymes et en utilisant les primitives `vp:onViewpoint` et `vp:on-Class`

Les méthodes proposées dans cette section permettent donc de construire des ontologies multi-points de vue à partir des ontologies classiques (sans point de vue) déjà existantes. Le résultat est une ontologie multi-points de vue représentée dans le modèle multi-points de vue `MVP` et dans langage d'ontologie multi-points de vue `MVP-OWL`.

5.4.3 Recherche la base d'annotations multi-points de vue selon un point de vue

Le modèle multi-points de vue `MVP` et le langage d'ontologie multi-points de vue `MVP-OWL` permettent, en utilisant une ontologie multi-points de vue, d'annoter des ressources dans le domaine d'intérêt selon plusieurs points de vue différents. Les annotateurs du système peuvent créer des énoncés sous forme des quadruplets $Q = <I_S, I_P, I_O, VP>$ où `VP` est un point de vue défini dans l'ontologie et $<I_S, I_P, I_O>$ est une annotation sous forme du triplet `RDF` à propos de la ressource I_S. L'annotation est considérée valide et vraie selon le point de vue `VP`. L'interprétation (sémantique) des annotations peut être contradictoire si ces annotations sont définies selon des points de vue différents. Les points de vue des annotateurs peuvent être définis et ajoutés dans l'ontologie en employant l'*Algorithme 23*. Le système permet alors de filtrer des annotations selon certains points de vue qui sont considérés pertinents pour un utilisateur final quelconque du système. L'utilisateur peut aussi demander selon quel point de vue, une annotation est créée (émise). Cela permet aux utilisateurs finaux du système de récupérer des connaissances qui sont pertinentes pour eux, de les aider à la navigation dans la base d'annotations selon des points de vue.

De même, au niveau terminologie, l'utilisateur peut demander des informations concernant le modèle de connaissances (ontologie) : par exemple, quelles sont les sous-classes d'une classe (la décomposition d'une classe), quelles sont les sous-classes d'une classe *selon* un point de vue particulier, une classe est une sous-classe d'une autre classe selon quel point de vue, deux classes sont-elles reliées par une passerelle ?…

Comme les ontologies en MVP-OWL correspondent aux ensembles de triplets RDF, les ontologies en MVP-OWL sont donc des graphes RDF. Alors, la recherche dans ce graphe RDF peut être effectuée par le moteur de recherche sémantique Corese [Corby et Faron, 2002].

5.5 Raisonnement avec MVP-OWL

Les mécanismes de raisonnement qui sont applicables sur les ontologies en OWL sont aussi applicables sur les ontologies en MVP-OWL, à condition qu'ils soient effectués en prenant en compte la notion de point de vue dans les cas nécessaires. [Bechhofer, 2003] présente quelques types d'inférence sur des classes et des instances dans des ontologies en OWL. Examinons l'exemple suivant à propos de l'inférence de classe :

```
:Bus_Driver
    a owl:Class;
    owl:equivalentClass
        [ a       owl:Class ;
          owl:intersectionOf  (
              :Person
              [ a owl:Restriction;
                owl:someValuesFrom :Bus;
                owl:onProperty :drive ]
                              )
        ] .
:Driver
    a owl:Class;
    owl:equivalentClass
        [ a       owl:Class ;
          owl:intersectionOf  (
              :Person
              [ a owl:Restriction;
                owl:someValuesFrom :Vehicle;
                owl:onProperty :drive ]
                              )
        ] .
:Bus
    a rdfs:Class;
    rdfs:subClassOf :Vehicle .
```

Cet extrait d'une ontologie en format N3 exprime trois connaissances :

- Bus_Driver est une Person qui conduit un Bus
- Driver est une Person qui conduit un Vehicle
- Bus est un Vehicle

Alors, l'inférence sur des classes nous permet de déduire une nouvelle connaissance : Bus_Driver est un Driver.

L'inférence peut aussi s'effectuer sur des instances. Prenons l'exemple suivant :

```
:Spike
    a owl:Thing;
    :isPetOf    :Pete .

:Pete
    a owl:Thing .

:hasPet
    a           owl:ObjectProperty ;
```

```
    rdfs:domain :Person ;
    rdfs:range  :Animal .

:isPetOf
    a           owl:ObjectProperty ;
    owl:inverseOf :hasPet .
```

Les connaissances exprimées dans cet extrait :

- `Spike` a une relation `isPetOf` avec `Pete`
- `Pete` est quelque chose
- `hasPet` est une relation entre `Person` et `Animal`
- `isPetOf` est une relation inverse de la relation `hasPet`

Alors, l'inférence sur des instances nous permet de déduire de nouvelles connaissances : `Pete` est une `Person`, `Spike` est un `Animal` et `Pete` a une relation `hasPet` avec `Spike`.

Ces types d'inférence sont aussi utilisables pour déduire des nouvelles connaissances dans les ontologies en `MVP-OWL`.

Pour les ontologies en `MVP-OWL`, grâce aux passerelles entre des entités (classes, propriétés, instances), d'autres types d'inférence peuvent aussi être utiles et utilisables. Ces nouveaux types d'inférence se basent aussi sur le principe de la *propagation des relations* comme ceux utilisés pour les ontologies en `OWL`. Deux types principaux de propagation des relations sont la *subsomption* et l'*instanciation*. Le premier exemple ci-dessus est basé sur la *propagation de la subsomption* entre des classes. Le deuxième exemple est basé sur la *propagation de l'instanciation*.

Nous présentons ici deux nouveaux types d'inférence via les passerelles entre des entités pour les ontologies multi-points de vue en langage `MVP-OWL`. Le premier type d'inférence se base sur la *propagation de la subsomption* et le deuxième sur la *propagation de l'instanciation*.

5.5.1 Propagation de la subsomption en MVP-OWL

Ce type d'inférence permet de déduire des nouvelles relations de subsomption entre des classes ou des propriétés à partir des relations de subsomption définies dans un point de vue et des passerelles entre des points de vue.

Si (1) nous avons deux classes A_1 et B_1 ; (2) la classe C_2 est définie comme sous-classe de la classe D_2 selon le point de vue `vp` ; et (3) il y a deux passerelles d'inclusion entre des classes : A_1 est incluse dans C_2 (A_1 est sous-classe de C_2) et B_1 inclut D_2 (D_2 est sous-classe de B_1) ; alors, nous pouvons déduire que la classe A_1 est sous-classe de la classe B_1.

La *Figure 60* montre un exemple de ce type d'inférence. `Véhicule` et `Camionnette` sont deux classes définies sans avoir une relation particulière entre elles. Ces classes sont liées aux classes `Voiture` et `Camion` par des passerelles d'inclusion.

`Camion` est définie comme sous-classe de `Voiture` selon le point de vue `vp`. Alors, nous pouvons déduire qu'il existe une relation de subsomption entre les deux classes `Véhicule` et `Camionnette` : la classe `Camionnette` est sous-classe de la classe `Véhicule`. Notons que cette nouvelle relation de subsomption est considérée comme valide selon le point de vue général VP_G.

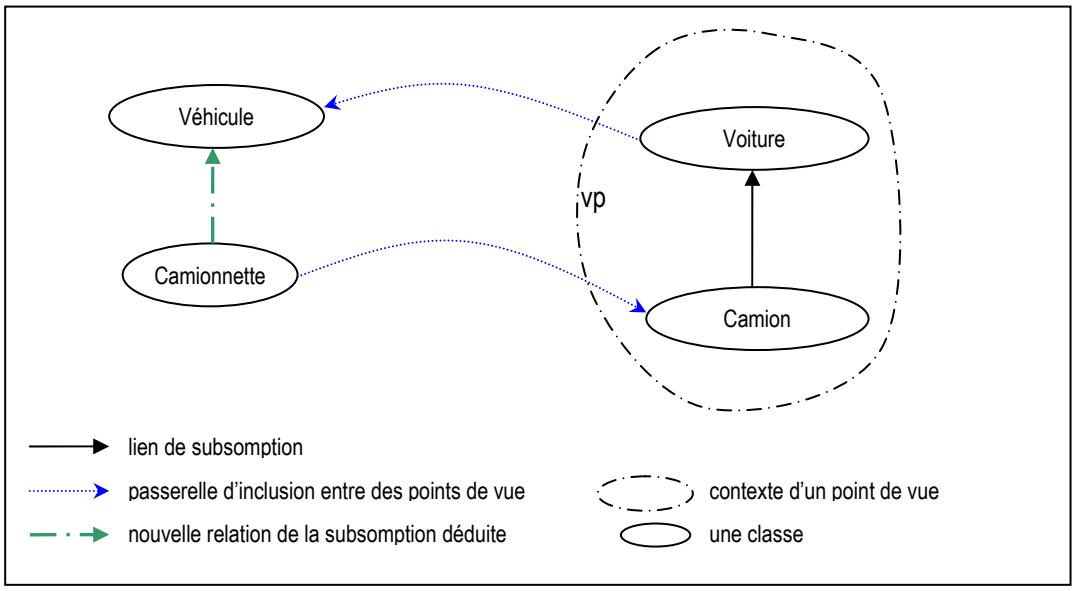


Figure 60 Propagation de la subsomption en MVP-OWL

Notons qu'à part des passerelles du type d'inclusion, les passerelles du type d'équivalence sont aussi utilisables pour ce type de déduction.

5.5.2 Propagation de l'instanciation en MVP-OWL

Ce type d'inférence permet de déduire de nouvelles relations d'instanciation.

Si (1) nous définissons une classe A_1 et une instance a_1 ; (2) nous décrivons l'instance b_2 est une instance de la classe B_2 selon le point de vue `vp` ; et (3) il y a deux passerelles : une passerelle d'inclusion entre deux classes A_1 et B_2 : A_1 inclut B_2 (B_2 est sous-classe de A_1) et une passerelle d'équivalence entre deux instances a_1 et b_2 indiquant qu'elles sont le même individu ; alors, nous pouvons déduire que l'instance a_1 est une des instances de la classe A_1 *selon* le point de vue `vp`. Autrement dit, a_1 instancie la classe A_1.

La *Figure 61* montre un exemple de ce type d'inférence. `Véhicule` et `Voiture_de_Rose` sont deux entités (une classe et une instance), elles sont liées aux entités `Voiture` et `Voiture_Rose` par une passerelle d'inclusion et une passerelle d'équivalence, respectivement. Selon le point de vue `vp`, `Voiture_Rose` est une instance de la classe `Voiture`. Alors, nous pouvons déduire qu'il existe une relation d'instanciation entre deux entités `Véhicule` et `Voiture_de_Rose` : `Voiture_de_Rose` est une instance de la classe `Véhicule` *selon* le point de vue `vp`.

Analogiquement à la propagation de la subsomption, à part des passerelles du type d'inclusion, les passerelles du type d'équivalence sont aussi utilisables pour ce type de déduction.

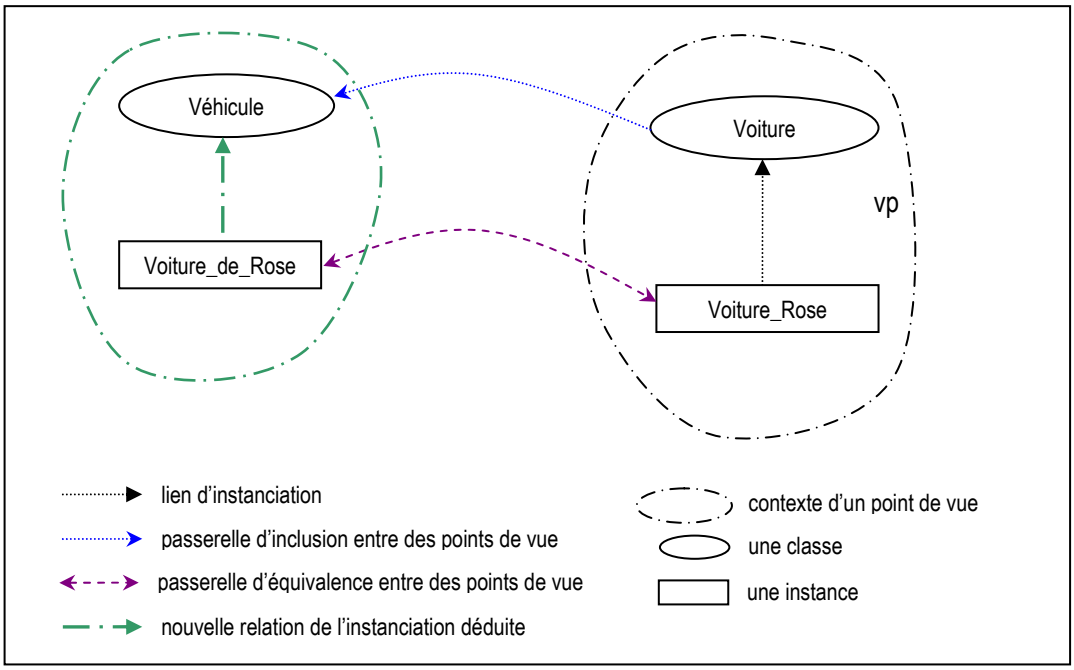


Figure 61 Propagation de l'instanciation en MVP-OWL

5.5.3 Raisonnement avec la passerelle d'exclusion

Ce type d'inférence permet de déduire si deux instances sont différentes.

Si (1) nous définissons deux classes `A`, `B` comme sous-classes d'une classe selon un point de vue ; (2) nous définissons aussi une instance `a` de la classe `A` selon le point de vue `vp` ; (3) il y a une passerelle d'exclusion entre deux classes `A` et `B`; alors, nous pouvons déduire que l'instance `a` de la classe `A` ne peut pas être une instance de la classe `B` selon le point de vue `vp`.

La *Figure 62* montre un exemple de ce type d'inférence. `Véhicule` est une classe. `Vélo` et `Voiture` sont deux sous-classes de la classe `Véhicule` selon le point de vue `vp`. `Vélo_Rose` est une instance de la classe `Vélo` selon le point de vue vp_1. La passerelle d'exclusion définie entre deux classes `Vélo` et `Voiture` permet de déduire que l'instance `Vélo_Rose` ne peut pas appartenir à l'ensemble d'instances de la classe `Voiture` selon le point de vue vp_1.

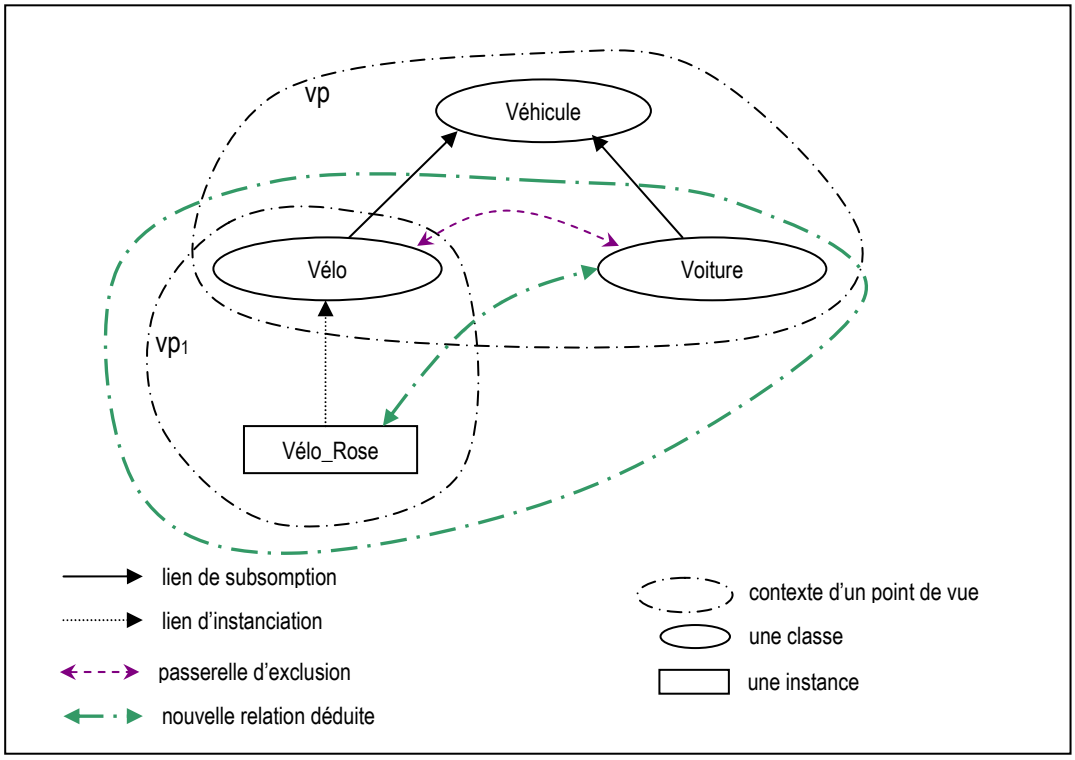


Figure 62 Raisonnement avec la passerelle d'exclusion en MVP-OWL

5.6 Conclusion

Dans ce chapitre, nous avons présenté notre modèle multi-points de vue. Ce modèle permet de représenter des connaissances dans un domaine, particulièrement dans une organisation ou une entreprise, en prenant en compte la notion de point de vue. Dans une grande organisation, qui peut être une entreprise unique ou une entreprise virtuelle constituée par plusieurs organisations en collaboration, il peut exister plusieurs communautés ou groupes des utilisateurs avec leurs propres points de vue. Ces points de vue dépendent du type de personne (métier, âge, niveau de formation, expérience, etc.) ou de l'utilisation (une même personne pourrait avoir différents points de vue en fonction de la tâche qu'elle cherche à accomplir). Le modèle multi-points de vue proposé permet donc de représenter des connaissances dans une entreprise, et accepte l'hétérogénéité avec des points de vue différents mais tout en gardant un certain niveau de la cohérence et de l'intégralité de ces connaissances dans le cadre de l'entreprise.

Notre modèle multi-points de vue, qui est différent d'autres modèles de représentation des connaissances multi-points de vue tels que C-Vista [Ribière, 1999] ou TROPES [Marino, 1993], permet d'exprimer le multi-héritage des classes selon différents points de vue. Par contre, comme C-Vista, notre modèle permet aussi de représenter une multi-représentation de l'objet réel (individu) et d'exprimer les passerelles entre des entités définies selon des points de vue différents. Cependant, au niveau terminologique, le modèle requiert une cohérence entre des interprétations des définitions d'une classe, si cette dernière est définie selon des points de vue différents. Cela assure une cohérence du modèle de connaissances au niveau de l'entreprise. Au niveau assertionnel, on peut énoncer des annotations à propos d'un ou plusieurs individus (objets réels) qui sont compatibles ou contradictoires. Cela permet aux utilisateurs ou aux annotateurs de l'organisation d'exprimer leurs connaissances à propos des objets dans le monde réel selon leurs points de vue.

En se basant sur le modèle multi-points de vue proposé, nous avons présenté une extension du langage d'ontologie `OWL`, nommé langage d'ontologie multi-points de vue `MVP-OWL`. Nous avons présenté la syntaxe abstraite et la sémantique de `MVP-OWL` en nous inspirant les travaux de W3C [Patel-Schneider *et al.*, 2004]. En introduisant 12 nouvelles primitives (dont une primitive d'instance, quatre primitives de classe et sept primitives de propriété, voir le *Tableau 19*) avec leur propre sémantique, et la redéfinition de la sémantique des constructeurs de `OWL`, `MVP-OWL` permet de représenter les ontologies multi-points de vue dans un langage similaire à `OWL`. Au niveau syntaxique, `MVP-OWL` et `OWL` sont assez proches. La façon d'écrire les descriptions à propos des entités (classes, propriétés, instances) dans l'ontologie n'est pas changée syntaxiquement. Cependant, l'interprétation des définitions des classes et des individus prend en compte la notion de point de vue via les primitives `vp:ClassWithViewpoint`, `vp:onClass`, `vp:onViewpoint` et `vp:belongToViewpoint`.

Un autre avantage du modèle multi-points de vue et du langage `MVP-OWL` est que la conversion d'une ontologie `OWL` vers une ontologie `MVP-OWL` est simple. La conversion est effectuée en définissant le point de vue de l'auteur de l'ontologie et en modifiant la définition d'une classe avec primitive `rdfs:subClassOf` par une autre définition avec

les nouvelles primitives `vp:ClassWithViewpoint`, `vp:onClass`, `vp:onViewpoint`. Pour les descriptions des individus, la modification est effectuée en employant la primitive `vp:belongToViewpoint` avec le point de vue venant d'être défini. Cela permet de contextualiser les descriptions des classes et des individus selon le point de vue, et de ne pas changer les significations des entités et les connaissances définies dans le domaine. Au lieu de définir le point de vue de l'auteur de l'ontologie, nous pouvons utiliser le point de vue général pour la conversion d'une ontologie en `OWL` vers celle en `MVP-OWL`. En plus, les parseurs qui sont « non-compatibles » avec `MVP-OWL` peuvent ignorer des primitives commençant par `vp` (`vp:belongToViewpoint`...) pour obtenir une « propre » ontologie en `OWL`.

En comparaison avec d'autres approches de représentation d'ontologie en prenant en compte la notion de point de vue, telle que `C-OWL` [Bouquet *et al*., 2003] [Bouquet *et al*., 2004] où une ontologie correspond à un contexte et il peut avoir des contradictions dans les définitions des entités dans une même ontologie, notre approche est la seule approche permettant de faire cohabiter entre l'hétérogénéité (au niveau assertionnel) et le consensus (au niveau terminologique). En plus, dans l'approche de `C-OWL`, les ontologistes ayant points de vue différents doivent modéliser le domaine de discours par des ontologies différentes (correspondant aux contextes ou aux points de vue différents) et les contextes/points de vue ne peuvent pas être caractérisés. Dans notre approche, les points de vue sont les entités dans le modèle `MVP-OWL`, cela permet donc de définir les points de vue autant que l'on veut et de modéliser le domaine de discours de manière la plus appropriée, par une seule ontologie multi-points de vue. Les points de vue peuvent aussi être caractérisés par des attributs (propriétés) des classes ou par des annotations pour spécifier sur quels critères des sous-classes d'une classe sont décomposées. Par ailleurs, une classe qui est définie comme sous-classe d'une autre classe selon un point de vue peut être décomposée selon d'autres points de vue pour avoir d'autres sous-classes. Ces dernières sont examinées par des attributs définis dans tous les points de vue où leurs classes ancestrales sont décomposées. `MVP-OWL` permet donc d'organiser les points de vue en hiérarchie, et cela n'est pas le cas de `C-OWL`.

Dans le chapitre suivant, nous allons présenter quelques exemples des définitions des entités dans une ontologie multi-points de vue en utilisant notre modèle de représentation des connaissances multi-points de vue `MVP` et notre langage d'ontologie multi-points de vue `MVP-OWL`.

Chapitre 6

Illustration du modèle MVP et du langage MVP-OWL

Dans ce chapitre, nous présentons une illustration du modèle de représentation des connaissances multi-points de vue MVP et du langage d'ontologie multi-points de vue MVP-OWL.

6.1 Illustration du modèle MVP et du langage MVP-OWL

L'illustration du modèle multi-points de vue MVP et du langage d'ontologie multi-points de vue MVP-OWL est effectuée par des exemples. Nous essayons de montrer des exemples où il est nécessaire d'employer la notion de « *point de vue* » et de montrer comment le modèle et le langage proposés répondent à ces demandes.

Le squelette d'une ontologie en MVP-OWL sérialisée en XML/RDF est montré dans la *Figure 63*. Comme OWL, une ontologie en MVP-OWL sérialisée en XML/RDF est représentée physiquement sous la forme d'un fichier texte structuré en éléments, à l'aide de balises éventuellement imbriquées. En en-tête du document figure un « prologue » : une déclaration identifiant le document comme un document XML, avec la version de XML et le codage de caractères employés pour ce fichier.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
   <!ENTITY xsd      "http://www.w3.org/2001/10/XMLSchema#">
   <!ENTITY rdf      "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
   <!ENTITY rdfs     "http://www.w3.org/2000/01/rdf-schema#" >
   <!ENTITY owl      "http://www.w3.org/2002/07/owl#">
   <!ENTITY vp       "http://www-sop.inria.fr/ACACIA/ontologies/mvpowl.owl#">
   <!ENTITY ex       "http://www-sop.inria.fr/ACACIA/ontologies/exemple.owl#">
]>

<rdf:RDF
   xmlns:xsd    ="&xsd;"
   xmlns:rdf    ="&rdf;"
   xmlns:rdfs   ="&rdfs;"
   xmlns:owl    ="&owl;"
   xmlns:vp     ="&vp;"

   xmlns:ex     ="&ex;"
   xmlns        ="&ex;"
   xml:base     ="&ex;" >

<!-- Définitions des entités de l'ontologie -->


</rdf:RDF>
```

Figure 63 Squelette d'une ontologie en MVP-OWL

Des abréviations pour des URIs des entités dans l'ontologie MVP-OWL peuvent être déclarées en employant des définitions d'entité (ENTITY) dans la déclaration du type de document (DOCTYPE). Les espaces de noms utilisés dans l'ontologie sont déclarés dans la balise rdf:RDF :

- les quatre premières déclarations introduisent des types de données de XML Schema, des objets définis dans l'espace de noms de RDF, de RDF Schema et le vocabulaire de OWL.

- La cinquième déclaration réfère des entités définies dans l'espace de noms `http://www-sop.inria.fr/ACACIA/ontologies/mvpowl.owl#`, qui sont les entités du modèle multi-points de vue `MVP` telles que les classes point de vue `vp:Viewpoint`, `vp:ClassWithViewpoint`… les propriétés `vp:belongToViewpoint`, `vp:EQUIV`…

- Les sixième et septième décalarations identifient l'espace de noms associé à l'ontologie en cours de définition. La septième déclaration d'espace de noms indique à quelle ontologie se rapporter en cas d'utilisation de noms sans préfixe.

- La dernière déclaration identifie l'URI de base de l'ontologie courante (le fichier de l'ontologie).

Les définitions des entités dans une ontologie en `MVP-OWL` sont effectuées entre deux balises `<rdf:RDF>` et `</rdf:RDF>`.

6.1.1 Définition d'une classe en MVP-OWL

Une classe dans l'ontologie `MVP-OWL` est une entité de modèle multi-points de vue `MVP`. Elle peut être définie comme sous-classe d'une ou plusieurs autres classes selon un ou plusieurs points de vue. Les descriptions concernant la sous-classe selon un point de vue sont groupées en des blocs. Chaque bloc correspond à une définition d'une ressource anonyme, qui est une sous-classe de la classe mère, selon le point de vue associé à ce bloc. La définition de la classe à définir et celle de la ressource anonyme sont reliées par la relation de subsomption via la primitive `rdfs:subClassOf`.

Le squelette de définition d'une classe de l'ontologie en `MVP-OWL` est montré dans la *Figure 64*.

```
<owl:Class rdf:ID="ClassID">
    <rdfs:subClassOf rdf:parseType="Resource">
       <!-- décrire que la classe ClassID est sous-classe de la classe
            ClassID_1 selon le point de vue VP_1 -->
       <vp:onViewpoint rdf:resource=”VP_1” />
       <vp:onClass rdf:resource=”ClassID_1” />
    </rdfs:subClassOf>

   <!-- ... -->

    <rdfs:subClassOf rdf:parseType="Resource">
       <!-- décrire que la classe ClassID est sous-classe de la classe
            ClassID_n selon le point de vue VP_n -->
       <vp:onViewpoint rdf:resource=”VP_n” />
       <vp:onClass rdf:resource=”ClassID_n” />
    </rdfs:subClassOf>

   <!-- Autres descriptions de la classe -->

</owl:Class>
```

Figure 64 Squelette de définition d'une classe de l'ontologie en MVP-OWL

Tous les autres types de description d'une classe tels que les annotations (par les primitives `rdfs:label`, `rdfs:comment`,…), l'intersection (`owl: intersectionOf`), l'union (`owl:unionOf`), le complément (`owl:complementOf`), l'équivalence (`owl:equivalentClass`), l'énumération (`owl:oneOf`), la disjonction (`owl: disjointWith`), la restriction (`owl:Restriction`, `owl:allValueFrom`,…) sont effectués comme en `OWL`.

L'exemple suivant est une définition de la classe `Animal` en `MVP-OWL` sans point de vue. Dans ce cas, la définition est syntaxiquement et sémantiquement similaire à celle en `OWL`.

```
<owl:Class rdf:ID="Animal">
    <rdfs:label>Animal</rdfs:label>
    <rdfs:comment> This class of animals is illustrative of a number of
                   ontological idioms. </rdfs:comment>
</owl:Class>
```

L'exemple suivant est une définition plus complexe à propos d'une classe avec l'utilisation du héritage multiple. La classe `Furniture` est définie comme sous-classe de deux autres classes selon deux points de vue différents : le point de vue « utilisation » `vp_Utilisation` et le point de vue « décoration » `vp_Decoration`. Selon le premier point de vue, `Furniture` est une classe sous-typée par la classe `Object`. D'après le deuxième point de vue, `Furniture` est une sous-classe de la classe `Art_Product`. Nous voyons dans cet exemple, deux descriptions de sous-classe différentes selon deux points de vue à propos d'une même classe. Par contre, la condition de cohérence et de pertinence du modèle multi-points de vue `MVP` requiert que ces descriptions différentes soient cohérentes et non contradictoires au niveau d'interprétation sémantique de la classe.

```
<rdf:Description rdf:ID="Furniture">
    <rdfs:subClassOf rdf:parseType="Resource">
       <vp:onViewpoint rdf:resource="#vp_Utilisation" />
       <vp:onClass rdf:resource="#Object" />
    </rdfs:subClassOf>

    <rdfs:subClassOf rdf:parseType="Resource">
       <vp:onViewpoint rdf:resource="#vp_Decoration" />
       <vp:onClass rdf:resource="#Art_Product" />
    </rdfs:subClassOf>

    <rdfs:label>Furniture</rdfs:label>
    <rdfs:comment> the movable objects which support the human body (seating
                   furniture and beds), provide storage, and hold objects on
                   horizontal surfaces above the ground. </rdfs:comment>
    <rdfs:comment> a product of artistic design and is considered a form of
                   decorative art. </rdfs:comment>
</rdf:Description>
```

6.1.2 Définition d'une propriété en MVP-OWL

Comme les classes en `MVP-OWL`, une propriété dans l'ontologie `MVP-OWL` est également une entité de modèle multi-points de vue `MVP`. Par contre, les propriétés ne peuvent pas être définies sous un point de vue particulier. Autrement dit, toutes les propriétés en `MVP-OWL` sont considérées comme définies selon le point de vue général. Donc, la définition d'une propriété en `MVP-OWL` est exactement analogue à celle en `OWL`.

Le squelette de définition d'une propriété de l'ontologie en MVP-OWL est montré dans la *Figure 65*.

```
<owl:ObjectProperty rdf:ID="PropertyID">

    <!-- Descriptions de la propriété -->

</owl:ObjectProperty>
```

Figure 65 Squelette de définition d'une propriété de l'ontologie en MVP-OWL

Notons que le squelette dans la *Figure 65* est une définition d'une propriété d'objet en employant la primitive `owl:ObjectProperty`. Nous pouvons également définir des propriétés de type de données, des propriétés fonctionnelles… en remplaçant la primitive `owl:ObjectProperty` par les primitives correspondantes (e.g. `owl:DatatypeProperty`, `owl:FunctionalProperty`…).

L'exemple suivant est une définition de la propriété d'objet `hasWife` :

```
<owl:ObjectProperty rdf:ID="hasWife">
    <owl:equivalentProperty rdf:resource="#hasFemaleSpouse" />
    <rdfs:subPropertyOf rdf:resource="#hasSpouse" />

    <rdfs:domain rdf:resource="#Man" />
    <rdfs:range  rdf:resource="#Woman" />
</owl:ObjectProperty>
```

L'exemple suivant montre une autre façon de définir une propriété d'objet sans préciser au début la primitive `owl:ObjectProperty` en employant la primitive `rdf:Description`. Le type de la ressource `read` est alors déduit via le lien d'équivalence exprimé par la primitive `vp:EQUIV` et le type de la ressource anonyme qui est une propriété d'objet.

```
<rdf:Description rdf:ID="read">
    <vp:EQUIV rdf:parseType="Resource">
        <rdf:type rdf:resource=”&owl;ObjectProperty” />
        <rdfs:domain rdf:resource="#Man" />
        <rdfs:range rdf:resource="#Book" />
    </vp:EQUIV>
</rdf:Description>
```

6.1.3 Définition d'un point de vue en MVP-OWL

Un point de vue dans le modèle MVP-OWL est une entité du modèle. Le point de vue est employé pour fixer des critères, une vue perspective où l'on examine les attributs d'une classe, qui sont pertinents dans cette vue. En décrivant des individus, le point de vue est employé pour fixer un contexte où les descriptions à propos d'un individu sont valides et vraies.

Un point de vue est décrit en employant des propriétés d'annotation telles que `rdfs:label`, `rdfs:comment`, `rdfs:seeAlso`, `rdfs:isDefinedBy` et `owl:versionInfo` ou d'autres propriétés d'annotation définies par les utilisateurs (par exemple

`dc:creator`[1] peut être considérée comme une propriété d'annotation). Les propriétés employées pour caractériser la vue perspective d'un point de vue sont décrites par la primitive `vp:characterisedBy`.

La *Figure 66* montre le squelette de la définition d'un point de vue.

```
<vp:Viewpoint rdf:ID="ViewpointID">

    <!-- Descriptions (annotations) du point de vue -->

    <vp:characterisedBy rdf:resource="PropertyID_1" />
    <!-- ... -->
    <vp:characterisedBy rdf:resource="PropertyID_n" />

</vp:Viewpoint>
```

Figure 66 Squelette de définition d'un point de vue de l'ontologie en MVP-OWL

L'exemple suivant montre deux définitions du point de vue « sex » et du point de vue « evolution ».

```
<vp:Viewpoint rdf:ID="vp_sex">
    <rdfs:label>Point de vue de sexe</rdfs:label>
    <rdfs:comment> Dans ce point de vue, on distingue des choses selon leurs sexes
    </rdfs:comment>
    <vp:characteriseBy rdf:resource=”#hasSex” />
</vp:Viewpoint>


<vp:Viewpoint rdf:ID="vp_evolution">
    <rdfs:label>Point de vue d'evolution</rdfs:label>
    <rdfs:comment> Dans ce point de vue, on distingue des choses selon leur âge
    </rdfs:comment>
    <vp:characteriseBy rdf:resource=”#hasAge” />
</vp:Viewpoint>
```

6.1.4 Liens entre points de vue en MVP-OWL

Les liens entre points de vue, tels que le lien d'équivalence (`vp:EQUIV`), le lien d'inclusion (`vp:INCL`) et le lien d'exclusion (`vp:EXCL`), entre deux classes définies selon deux points de vue distincts permettent d'identifier le rapport entre des significations des classes. L'exemple suivant est une représentation d'un extrait de l'ontologie « Accidentologie » [Ribière, 1999] (voir *Figure 67*) dans le langage d'ontologie multi-points de vue `MVP-OWL` :

```
<vp:Viewpoint rdf:ID="vp_Véhicule">
    <rdfs:label>Point de vue de véhicule</rdfs:label>
    <rdfs:comment>
       Contexte de point de vue : Analyse Accident
       Objectif de point de vue : Analyse véhicule comme Facteur
       Expert (auteur) de point de vue : Joel
       Domaine de point de vue  : Infrastructure
    </rdfs:comment>
</vp:Viewpoint>
```

[1] http://purl.org/dc/elements/1.1/

```
<vp:Viewpoint rdf:ID="vp_Infrastructure">
    <rdfs:label>Point de vue de infrastructure</rdfs:label>
    <rdfs:comment>
       Contexte de point de vue : Analyse Accident
       Objectif de point de vue : Analyse infra comme Facteur
       Expert (auteur) de point de vue : Joel
       Domaine de point de vue  : Infrastructure
    </rdfs:comment>
</vp:Viewpoint>

<vp:Viewpoint rdf:ID="vp_Conducteur">
    <rdfs:label>Point de vue de conducteur</rdfs:label>
    <rdfs:comment>
       Contexte de point de vue : Analyse Accident
       Objectif de point de vue : Analyse conducteur comme Facteur
       Expert (auteur) de point de vue : Yves
       Domaine de point de vue  : Psychologie du Conducteur
    </rdfs:comment>
</vp:Viewpoint>

<owl:Class rdf:ID="Facteur_Accident">
    <rdfs:label> Facteur Accident </rdfs:label>
</owl:Class>

<owl:Class rdf:ID="Pneu_Sous_Gonflé">
    <rdfs:subClassOf rdf:parseType="Resource">
       <vp:onViewpoint rdf:resource=”vp_Véhicule” />
       <vp:onClass rdf:resource=”#Facteur_Accident” />
    </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="Inexistence_Signal_Spécifique_Sortie_Chaussée">
    <rdfs:subClassOf rdf:parseType="Resource">
       <vp:onViewpoint rdf:resource=”vp_Véhicule” />
       <vp:onClass rdf:resource=”#Facteur_Accident” />
    </rdfs:subClassOf>
    <vp:EQUIV rdf:resource=”#Ambiguité_Clignotant” />
</owl:Class>

<owl:Class rdf:ID="Ambiguité_Clignotant">
    <rdfs:subClassOf rdf:parseType="Resource">
       <vp:onViewpoint rdf:resource=”vp_Conducteur” />
       <vp:onClass rdf:resource=”#Facteur_Accident” />
    </rdfs:subClassOf>
    <vp:INCL rdf:resource=”#Conflit_Dépassement” />
    <vp:INCL rdf:resource=”#Conflit_Tourne_à_Gauche” />
</owl:Class>

<owl:Class rdf:ID="Règle_à_Adopter">
    <rdfs:subClassOf rdf:parseType="Resource">
       <vp:onViewpoint rdf:resource=”vp_Conducteur” />
       <vp:onClass rdf:resource=”#Facteur_Accident” />
    </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="Conflit_Dépassement">
    <rdfs:subClassOf rdf:parseType="Resource">
       <vp:onViewpoint rdf:resource=”vp_Infrastructure” />
       <vp:onClass rdf:resource=”#Facteur_Accident” />
    </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="Conflit_Tourne_à_Gauche">
    <rdfs:subClassOf rdf:parseType="Resource">
       <vp:onViewpoint rdf:resource=”vp_Infrastructure” />
       <vp:onClass rdf:resource=”#Facteur_Accident” />
    </rdfs:subClassOf>
</owl:Class>
```

```
<owl:Class rdf:ID="Route_3_voies">
    <rdfs:subClassOf rdf:parseType="Resource">
       <vp:onViewpoint rdf:resource=”vp_Infrastructure” />
       <vp:onClass rdf:resource=”#Facteur_Accident” />
    </rdfs:subClassOf>
</owl:Class>
```

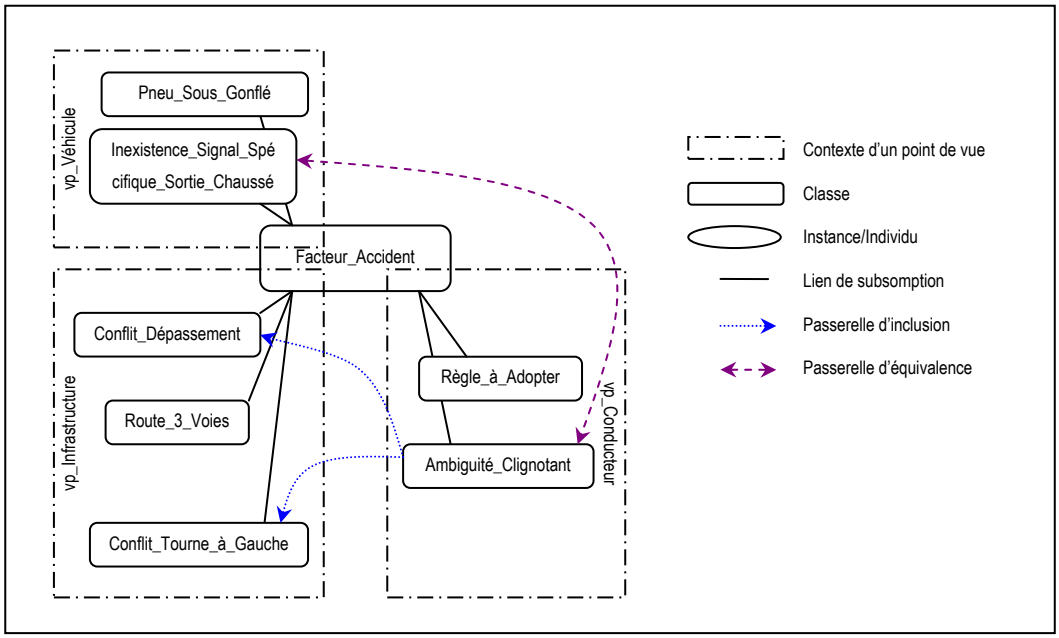


Figure 67 Exemple de liens entre points de vue

6.1.5 Annotations et faits

Contrairement au niveau terminologique, où le modèle multi-points de vue `MVP` exige de la cohérence, du consensus et ne permet pas d'avoir de contradiction dans l'interprétation des descriptions des classes avec les points de vue, au niveau assertionnel, nous pouvons émettre des descriptions différentes ou des annotations différentes à propos des objets réels (individus) selon différents points de vue. Les annotateurs ou les utilisateurs finaux donc peuvent faire des énoncés (annotations) comme ils veulent et selon leurs connaissances et leurs points de vue.

Le squelette en `MVP-OWL` pour décrire un objet réel (individu) est montré dans la *Figure 68*. Dans chaque bloc `<vp:EQUIV>` … `</vp:EQUIV>`, nous écrivons un ensemble de descriptions à propos de l'objet réel selon un point de vue particulier. Les descriptions dans un point de vue doivent être cohérentes mais elles peuvent être contradictoires avec celles décrites dans un autre point de vue. D'autres descriptions peuvent être écrites dehors les blocs `<vp:EQUIV>` … `</vp:EQUIV>`, dans ce cas, elles sont considérées comme décrites selon le point de vue général.

```
<owl:Thing rdf:ID="ObjectID">
    <vp:EQUIV rdf:parseType="Resource">
       <vp:belongToViewpoint rdf:resource=”VP_1” />

       <!-- Descriptions de l’objet réel selon le point de vue VP_1 -->

    </vp:EQUIV>

   <!-- ... -->

    <vp:EQUIV rdf:parseType="Resource">
       <vp:belongToViewpoint rdf:resource=”VP_n” />

       <!-- Descriptions de l’objet réel selon le point de vue VP_n -->

    </vp:EQUIV>

    <!-- Autres descriptions de l’objet réel, ces descriptions sont considérées
         comme décrites selon le point de vue général -->

</owl:Thing>
```

Figure 68 Squelette de description d’un individu (objet réel) en MVP-OWL

L’exemple suivant est une description de la voiture de Rose, qui est un objet réel référencé par l’URI `http://www-sop.inria.fr/ACACIA/ontologies/exemple.owl#Voiture_de_Rose`. La voiture est décrite différemment selon le point de vue de Khaled et celui de Sylvain. Selon Khaled, il voit que la voiture de Rose a 5 portes et a la couleur verte mais d’après Sylvain, il considère que cette voiture n’a que 4 portes et elle est verte claire. Par contre, tout le monde est d’accord sur le fait que la voiture de Rose a 4 pneus.

```
<owl:Thing rdf:ID="Voiture_de_Rose">
    <vp:EQUIV rdf:parseType="Resource">
       <vp:belongToViewpoint rdf:resource=”#vp_Khaled” />
       <rdf:type rdf:resource=”&owl;Thing” />
       <rdf:type rdf:resource=”#Voiture” />
       <rdfs:label>Voiture de Rose</rdfs:label>
       <aCouleur rdf:resource=”#Couleur_Verte” />
       <nombrePortes rdf:datatype="&xsd;nonNegativeInteger"> 5 </nombrePortes>
    </vp:EQUIV>

    <vp:EQUIV rdf:parseType="Resource">
       <vp:belongToViewpoint rdf:resource=”#vp_Sylvain” />
       <rdf:type rdf:resource=”&owl;Thing” />
       <rdf:type rdf:resource=”#Voiture” />
       <rdfs:label>Véhicule personnel de Rose</rdfs:label>
       <rdfs:comment> Je trouve que la voiture de Rose est verte claire.
                      </rdfs:comment>
       <aCouleur rdf:resource=”#Couleur_Verte_Claire” />
       <nombrePortes rdf:datatype="&xsd;nonNegativeInteger"> 4 </nombrePortes>
    </vp:EQUIV>

    <nombrePneus rdf:datatype="&xsd;nonNegativeInteger"> 4 </nombrePneus>
</owl:Thing>
```

Notons que les descriptions d’un objet réel sont également considérées comme des annotations. Dans un système employant le modèle multi-points de vue `MVP`, les annotations sont stockées dans une base d’annotations, sous forme des quadruplets <I_S, I_P, I_O, `VP`>. Par exemple, <`Voiture_de_Rose`, `aCouleur`, `Couleur_Verte`, `vp_Khaled`> est une annotation stockée dans la base d’annotations.

6.2 Conclusion

Dans ce chapitre, nous avons présenté une illustration d'utilisation du modèle multi-points de vue MVP et du langage d'ontologie multi-points de vue MVP-OWL, qui est effectuée via des exemples des définitions des entités de l'ontologie multi-points de vue, sérialisées en format XML/RDF. Les squelettes et les exemples concrets montrent que le modèle MVP et le langage MVP-OWL permettent de définir des entités du modèle ou de l'ontologie telles que des classes, des propriétés, des instances en prenant en compte de la notion de « *point de vue* ». La définition de la relation de subsomption entre deux classes peut être effectuée selon un ou plusieurs points de vue, ou selon le point de vue général. Les descriptions des individus peuvent aussi être effectuées dans un ou plusieurs points de vue. Cela permet à des ontologistes différents, des groupes différents dans une organisation de définir des entités de l'ontologie de la manière la plus pertinente et convenable pour eux en assurant toujours un certain niveau de la cohérence et du consensus généraux du modèle (de l'ontologie).

Un point de vue est défini et employé pour caractériser et fixer un contexte, une condition, des critères où certains attributs d'une classe définie dans l'ontologie multi-points de vue sont considérés comme valides, valables et pertinents.

Conclusion et Perspectives

Le *Web sémantique* est la vision du Web de demain, où nous pouvons effectuer plusieurs tâches de manière informatisée et automatisée grâce à l'explicitation de la sémantique des ressources et des connaissances, explicitation les rendant « compréhensibles » et manipulables par les machines et les logiciels. Dans le noyau du Web sémantique, les *ontologies* permettent de capitaliser, de représenter, d'exploiter et de partager sémantiquement des connaissances et des informations. En tant qu'entités du Web, un environnement très large, très complexe et divers, les entités du Web sémantique possèdent aussi des caractéristiques de cet environnement si *hétérogène*. Les ontologies du Web sémantique peuvent donc être variées et hétérogènes même si elles sont créées dans un même domaine. Une des clés importantes pour le succès du Web sémantique est de maîtriser cette hétérogénéité et de cohabiter avec elle.

Pour atteindre l'objectif de la thèse qui est de permettre la construction et l'exploitation d'un Web sémantique dans une organisation hétérogène, nous envisageons la problématique de l'hétérogénéité de l'ontologie, qui est une composante de base importante du Web sémantique. Notre approche pour répondre à cette problématique a été la suivante :

- Étudier et proposer des algorithmes permettant de *mettre en correspondance automatiquement les ontologies*. Bien que l'ontologie soit utilisée pour capitaliser des connaissances d'un domaine, du fait de l'ouverture du Web et donc du Web sémantique, différentes communautés de même domaine d'intérêt peuvent utiliser différentes ontologies. L'objectif de nos algorithmes est de permettre donc à des communautés d'échanger, de partager des données, des informations entre elles. Par exemple, les connaissances dans une communauté (sous forme des annotations des documents, des ressources reposant sur l'ontologie utilisée par la communauté) deviennent compréhensibles pour une autre communauté qui emploie une autre ontologie, si leurs ontologies sont mises en correspondance, et si pour chaque concept d'une ontologie, l'on peut trouver un concept correspondant dans l'autre ontologie. Nos algorithmes résolvent le problème d'aligner automatiquement les ontologies représentées dans deux formalismes particuliers recommandés par W3C : l'algorithme ASCO1 pour des ontologies en RDF(S), les algorithmes ASCO2 et ASCO3 pour des ontologies en OWL.

- Étudier et proposer un modèle de représentation des connaissances en prenant en compte la notion de point de vue : *modèle multi-points de vue*. Ensuite, l'application de ce modèle pour construire une *ontologie multi-points de vue* dans une organisation hétérogène. Dans une organisation hétérogène se composant de plusieurs communautés, de plusieurs groupes d'utilisateurs qui sont répartis dans plusieurs différents endroits, différentes villes voire différents pays, qui ont des cultures diverses, différentes expériences, des niveaux de formation variés…, la maintenance d'un consensus au niveau global de l'organisation est très difficile voire impossible. Le modèle multi-points de vue et l'ontologie multi-points de vue proposés visent à exprimer le consensus et l'hétérogénéité dans telle organisation.

Contributions

De façon générale, les travaux effectués dans cette thèse fournissent une aide à la construction et l'exploitation d'un Web sémantique dans un environnement hétérogène en proposant des moyens permettant de travailler et cohabiter avec l'hétérogénéité dans le Web sémantique.

Nous précisons dans cette contribution générale, différentes contributions portant sur des aspects plus spécifiques de notre approche :

Contribution dans la tâche de mettre en correspondance des ontologies

La tâche de mise en correspondance des ontologies, ou l'alignement des ontologies, a émergé dans les dernières années depuis l'avènement du Web sémantique. Le résultat de cette tâche assure l'échange, le partage, la diffusion des données et des informations entre des systèmes ou des communautés dans le Web sémantique. Nous avons proposé trois algorithmes qui attaquent cette importante tâche.

L'algorithme ASCO1 vise à chercher des correspondances entre deux ontologies représentées en `RDF(S)`. Il exploite des caractéristiques du langage `RDF(S)` pour déduire la similarité entre deux entités de deux ontologies. La valeur de similarité est calculée en deux étapes : l'étape linguistique qui repose sur le nom, des étiquettes, des commentaires de l'entité, et l'étape structurelle qui se base sur des rapports entre des entités et leurs places dans les taxonomies d'entité. L'algorithme ASCO1 est un des premiers algorithmes dans la famille des algorithmes alignant des ontologies représentées en `RDF(S)`. C'est aussi la seule approche qui applique la technique très efficace et performante de classification des documents dans le domaine de recherche d'information : la technique de TF/IDF, pour la tâche d'alignement des ontologies. ASCO1 emploie également WordNet, un système de référence lexicologique [Miller, 1995], ainsi que des heuristiques dans les structures des ontologies pour augmenter la précision de l'algorithme.

Les algorithmes ASCO2 et ASCO3 visent à chercher des alignements entre deux ontologies représentées en `OWL`, le langage d'ontologie recommandé par W3C pour le

Web sémantique. Ces algorithmes essaient d'exploiter les avantages d'expressivité du langage d'ontologie OWL pour déduire la similarité de deux entités dans les ontologies en OWL. ASCO2 prend en compte principalement les informations dans la *structure locale* des entités des ontologies (ces informations sont contenues dans les descriptions des entités). La combinaison des similarités partielles obtenues grâce à la comparaison des composantes dans des définitions des entités est effectuée de manière originale : il s'agit d'une combinaison avec le *schéma de pondération variable*. Cette méthode montre un résultat encourageant dans la déduction de similarités entre des entités.

L'algorithme ASCO3, quant à lui, analyse les rapports sémantiques entre des entités dans des ontologies (*structure globale*) pour extraire des correspondances entre des ontologies. ASCO3 est la seule approche appliquant, pour l'alignement des ontologies, la notion d'*isomorphisme des graphes* et les algorithmes de recherche de sous-graphe commun de deux graphes, qui sont bien étudiés dans la théorie des graphes. Son idée est de considérer des ontologies en OWL, définies en employant les primitives de OWL qui ont les sémantiques bien prédéfinies, comme des *réseaux sémantiques* dont les nœuds sont les entités, les arcs sont les primitives de OWL. Ces réseaux sémantiques reflètent la conceptualisation et la modélisation des ontologies, et donc, plus ils sont proches, plus les ontologies sont similaires. Ainsi, ASCO3 cherche les *sous-graphes communs les plus larges* de ces réseaux en prenant en compte la sémantique des arcs en autorisant le « *saut* » d'un nœud à un autre, pour déduire des correspondances (les entités des ontologies correspondent aux nœuds des sous-graphes communs maximums trouvés) entre deux ontologies.

L'évaluation des algorithmes d'alignement proposés montre qu'en général, les résultats obtenus par l'algorithme ASCO2 sont meilleurs que ceux renvoyés par ASCO1, et ASCO3 renvoie les meilleurs résultats. Pour les cas plus précis, où les ontologies à aligner sont plus similaires au niveau linguistique qu'au niveau structurel, ASCO1 est un meilleur choix. Pour les cas contraires, où la structure des ontologies sont proches, ASCO2 et ASCO3 sont plus efficaces. En comparaison avec d'autres algorithmes en exécutant les mêmes tests dans les campagnes d'évaluation des outils d'alignement d'ontologies, nos algorithmes montrent des résultats probants et encourageants.

Contribution pour le modèle multi-points de vue et pour le langage d'ontologie multi-points de vue

Toujours pour le but de s'adapter à l'hétérogénéité dans le Web sémantique, nous proposons un modèle de représentation des connaissances multi-points de vue, le *modèle multi-points de vue*. Le modèle proposé permet d'avoir plusieurs définitions différentes à propos d'une entité, au niveau de la conceptualisation (niveau terminologique) mais en assurant toujours le consensus entier du modèle dans le cadre d'une organisation ou d'une entreprise. Au niveau assertionnel (des faits, des connaissances, des annotations), les utilisateurs du système peuvent définir leurs points de vue identiques et décrire leurs propres énoncés à propos des objets réels, qui peuvent être consensuels ou contradictoires.

En se basant sur le modèle multi-points de vue proposé, un langage d'ontologies multi-points de vue est construit, nommé le *langage d'ontologie multi-points de vue* MVP-

OWL. Ce langage permet de représenter des ontologies multi-points de vue selon le modèle multi-points de vue par un langage qui est une extension du langage d'ontologie OWL, recommandé par W3C. C'est le premier langage permettant de représenter des ontologies multi-points de vue basé sur OWL, tout en gardant la syntaxe de OWL et les primitives de OWL. Cependant, la sémantique de certaines constructions de OWL est modifiée et les nouvelles primitives sont ajoutées pour s'adapter avec le modèle multi-points de vue. Les opérateurs de base permettant de manipuler avec le langage MVP-OWL sont aussi étudiés. Enfin, nous proposons une méthode de construction d'une ontologie multi-points de vue en MVP-OWL à partir des ontologies déjà existantes, en utilisant des résultats de la tâche d'alignement des ontologies. Cela encourage l'application et l'implémentation du modèle et du langage proposés dans le Web sémantique en général et dans le Web sémantique d'entreprise en particulier.

Limites et Perspectives

- Les algorithmes proposés ont été implémentés en Java pour évaluer la qualité des correspondances entre deux ontologies trouvées. Cependant, ils ne sont pas encore optimisés au niveau de la performance, en particulier, pour l'algorithme ASCO3, qui applique l'algorithme de recherche de la clique maximale dans le graphe d'association de deux graphes représentant deux ontologies à comparer. Le problème de recherche de la clique maximale est un problème NP-complet et la solution est la recherche exhaustive avec des heuristiques. Il est nécessaire d'étudier des heuristiques particulières dans la structure de l'ontologie en OWL afin de réduire le temps de calcul pour la recherche de la clique maximale.

- Nous n'avons pu faire une validation du modèle multi-points de vue et du langage d'ontologie multi-points de vue dans le cadre d'une application concrète. Une validation de notre modèle et langage est envisagée, par exemple, dans le domaine médical.

- Une autre approche intéressante est d'étudier les relations entre des ontologies « classiques » et des ontologies multi-points de vue selon le modèle multi-points de vue proposé. Il est probablement intéressant de pouvoir détecter des correspondances entre des entités d'une ontologie traditionnelle et d'une ontologie multi-points de vue. Un scénario à considérer est celui d'une entreprise possédant déjà une ontologie multi-points de vue, et intégrant, au cours de sa croissance, d'autres communautés ou d'autres groupes d'utilisateurs qui possèdent eux-mêmes leurs propres ontologies. Le problème est donc de mettre en correspondance non seulement entre des ontologies classiques, mais aussi entre une ontologie multi-points de vue et une ontologie classique voire entre des ontologies multi-points de vue.

- Une perspective est la réalisation d'un système complet permettant d'une part de gérer des ontologies multi-points de vue par la définition, la modification, la suppression des points de vue, des entités de l'ontologie selon des points de vue ; et d'autre part de représenter des connaissances, de filtrer des connaissances

selon des points de vue des utilisateurs, de naviguer contextuellement dans la base des connaissances ou la mémoire d'entreprise selon les points de vue des utilisateurs, d'exploiter des points de vue des utilisateurs pour une présentation des résultats adaptée à l'utilisateur.

Bibliographie

[Bach *et al.*, 2004] Bach, T.L., Dieng-Kuntz, R. et Gandon, F. *On Ontology Matching Problems (for building a corporate Semantic Web in a multi-communities organization)*. Proceedings of ICEIS 2004. Porto, Portugal. 14-17 April 2004.

[Bach et Dieng, 2005] Bach, T.L. et Dieng-Kuntz, R. *Measuring Similarity of Elements in OWL Ontologies*. AAAI'2005 workshop on Contexts and Ontologies: Theory, Practice and Applications. Pittsburgh, Pennsylvania, USA, July 9, 2005, p.96-99.

[Bechhofer, 2003] Bechhofer S. *OWL Reasoning Examples*, http://owl.man.ac.uk/2003/why/latest/

[Benerecetti *et al*., 2001] Benerecetti, M., Bouquet, P., Ghidini, C. *On the Dimensions of Context Dependence: Partiality, Approximation, and Perspective*. Dans Proc. of CONTEXT 2001: 59-72

[Berners-Lee, 2000] Berners-Lee, T. *Semantic Web* - XML2000 http://www.w3.org/2000/ Talks/1206-xml2k-tbl/

[Berners-Lee *et al.*, 2001] Berners-Lee, T., Hendler, J., Lassila, O. *The semanticweb*. Sci Am 284(5):34–43, 2001.

[Berners-Lee, 2003] Berners-Lee, T. *WWW past & future*. Available at http://www.w3.org/2003/ Talks/0922-rsoc-tbl. Slide 30. 2003

[Bobrow et Winograd, 1977] Bobrow, D. G. et Winograd, T. *An overview of KRL, a knowledge representation language*. Cognitive Science, 1(1), 3--46. 1977.

[Bouquet *et al.*, 2003] Bouquet, P., Giunchiglia, F., van Harmelen, F., Serafini, L., et Stuckenschmidt, H. *C-OWL: Contextualizing Ontologies*. International Semantic Web Conference 2003: 164-179

[Bouquet *et al.*, 2004] Bouquet, P., Giunchiglia, F., van Harmelen, F., Serafini, L., et Stuckenschmidt, H. *Contextualizing Ontologies*. Dans Journal of Web Semantics, 2004.

[Bron et Kerbosch, 1973] Bron, C. et Kerbosch, J. *Finding All the Cliques in an Undirected Graph*. Communication of the Association for Computing Machinery 16, pp. 575-577,1973.

[Bunke *et al.*, 2002] Bunke, H., Foggia, P., Guidobaldi, C., Sansone, C. et Vento, M. *A Comparison of Algorithms for Maximum Common Subgraph on Randomly Connected Graphs*. SSPR/SPR Structural and Syntactic Pattern Recognition 2002: 123-132

[Castano *et al*., 2001] Castano, S., De Antonellis, V., et De Capitani di Vimercati, S. *Global viewing of heterogeneous data sources*. IEEE Transactions on Knowledge and Data Engineering, 13(2):277–297, 2001.

[Cohen *et al*., 2003] Cohen, W., Ravikumar, P., et Fienberg, S. *A comparison of string metrics for matching names and records.* Dans Proc. KDD-2003 Workshop on Data Cleaning and Object Consolidation, 2003.

[Corby *et al*., 2000] Corby, O., Dieng-Kuntz, R. et Hébert, C. *A conceptual graph model for w3c resource description framework.* In Proc. 8th International Conference on Conceptual Structures Logical, Linguistic, and Computational Issues, Darmstadt (DE), August 2000. Springer-Verlag.

[Corby *et al*., 2004] Corby, O., Dieng-Kuntz, R. et Faron-Zucker, C. *Querying the semantic web with the corese search engine.* Dans Proc. 15th ECAI/PAIS, Valencia (ES), August 2004. IOS Press.

[Corby et Faron, 2002] Corby, O. et Faron, C. *Corese: A corporate semantic web engine.* Dans proceedings of the International Workshop on Real World RDF and Semantic Web Applications, 11th International World Wide Web Conference, Hawai, USA, May, 2002.

[Corcho et Gomez-Perez, 2001] Corcho, O. et Gomez-Perez, A. *Solving Integration Problems of E-Commerce Standards and Initiatives through Ontological Mappings*. IJCAI-01 Workshop on Ontologies and Information Sharing, pages 131–140, 2001.

[Cox, 1994] Cox, T. et Cox, M. *Multidimensional Scaling.* Chapman and Hall, 1994.

[Dagan *et al.*, 1999] Dagan, I., Lee, L., et Pereira, F. *Similarity-based models of word cooccurrence probabilities*. Machine Learning 34(1-3).

[D'Aquin *et al*., 2003] D'Aquin, M., Bouthier, C., Brachais, S., Lieber, J., et Napoli, A. *Knowledge editing and maintenance tools for a semantic portal in ontology.* International Journal of Human-Computer Studies. Volume 62, Issue 5, Pages: 619 - 638 (May 2005).

[D'Aquin, 2005] D'Aquin, M. Thèse *Un portail sémantique pour la gestion des connaissances en cancérologie.* 2005.

[David, 1987] David, H. *Views : Multiple perspectives and structured objects in a knowledge representation language*. Master's thesis, MIT. 1987.

[Dekker, 1994] Dekker, L. *FROME : Représentation multiple et classification d'objets avec points de vue*. Thèse de doctorat, Université des Sciences et Technologies de Lille, Laboratoire d'Informatique Fondamentale de Lille, Juin 1994.

[Dieng et Hug, 1998a] Dieng, R. et Hug, S. *Comparison of "personal ontologies" represented through conceptual graphs.* Dans Proc. 13th ECAI, Brighton (UK), pages 341–345, 1998.

[Dieng et Hug, 1998b] Dieng, R. et Hug, S. *MULTIKAT, a Tool for Comparing Knowledge from Multiple Experts*. Dans Proc. of the 6th Int. Conference on Conceptual Structures (ICCS'98), Springer-Verlag, LNAI 1453. 1998.

[Dieng *et al.*, 2004] Dieng, R., Corby, O., Gandon, F. et Golebiowska, J. *Ontologies pour un Web sémantique d'entreprise*. Chapitre 1 de Gestion Dynamique des Connaissances Industrielles, Benoît Meynard, Muriel Lombard, Nada Matta, Jean Renaud, éds, Hermès, 2004.

[Do et Rahm, 2002] Do, H.H. et Rahm, E. *Coma – a system for flexible combination of schema matching approaches.* Dans Proc. VLDB, pages 610–621, 2002.

[Doan *et al*., 2000] Doan, A.H., Domingos, P.et Halevy, A. *Learning source descriptions for data integration*. In: ProcWebDBWorkshop, pp. 81–92, 2000.

[Doan et al., 2001] Doan, A.H., Domingos, P.et Halevy, A. *Reconciling schemas of disparate data sources: A machine-learning approach*. In Proceeding of SIGMOD, 2001.

[Doan *et al*., 2002] Doan, A., Madhavan, J., Domingos, P., et Halevy, A. *Learning to Map between Ontologies on the Semantic Web*. The 11th International World Wide Web Conference (WWW'2002), Hawaii, USA.

[Doan *et al*., 2003] Doan, A.H., Madhavan, J., Dhamankar, R., Domingos, P. et Halevy, A. *Learning to Match Ontologies on the Semantic Web.* VLDB journal, 2003.

[Domingos et Pazzani, 1997] Domingos, P. et Pazzani, M. *On the Optimality of the Simple Bayesian Classifier under Zero-One Loss*. Machine Learning, 29:103, 1997.

[Durand *et al.*, 1999] Durand, P. J., Pasari, R., Baker, J. W. et Tsai, C. *An Efficient Algorithm for Similarity Analysis of Molecules* . Internet Journal of Chemistry, vol. 2, 1999.

[Durban *et al*., 1998] Durban, R., Eddy, S. R., Krogh, A., et Mitchison, G. *Biological sequence analysis - Probabilistic models of proteins and nucleic acids*. Cambridge: Cambridge University Press, 1998.

[Ehrig et Staab, 2004] Ehrig, M. et Staab, S. *QOM - quick ontology mapping.* Dans Proc. 3rd ISWC, Hiroshima (JP), November 2004.

[Ehrig et Sure, 2004] Ehrig, M. et Sure, Y. *Ontology mapping – an integrated approach.* Dans Christoph Bussler, John Davis, Dieter Fensel, and Rudi Studer, editors, Proc. 1st ESWS,

Hersounisous (GR), volume 3053 of Lecture Notes in Computer Science, pages 76–91. Springer Verlag, MAY 2004.

[Euzenat *et al.*, 2004] Euzenat, J., Bach, T.L., Barrasa, J., Bouquet, P., Bo, J.D., Dieng-Kuntz, R., Ehrig, M., Hauswirth, M., Jarrar, M., Lara, R., Maynard, D., Napoli, A., Stamou, G., Stuckenschmidt, H., Shvaiko, P., Tessaris, S., Acker; S.V. et Zaihrayeu, I. *State of the art on ontology alignment*, deliverable 2.2.3, IST Knowledge web NoE, Knowledge web NoE, 80p., June 2004.

[Euzenat et Valtchev, 2004] Euzenat, J. et Valtchev, P. *Similarity-based ontology alignment in OWL-lite.* Dans Proc. 15th ECAI, Valencia (ES), 2004.

[Falquet et Mottaz Jiang, 2001] Falquet, G., Mottaz Jiang, C.L. *Navigation hypertexte dans une ontologie multi-points de vue*. Dans Proc. NimesTIC-01 conf., Nîmes, France, December 2001.

[Falquet et Mottaz Jiang, 2002] Falquet, G., Mottaz Jiang, C.L. *A Model for the Collaborative Design of Multi-Point-of-View Terminological Knowledge Bases*. Dans R. Dieng and N. Matta (Eds) Knowledge Management and Organizational Memories, Kluwer, 2002.

[Fellegi et Sunter, 1969] Fellegi, I. P. et Sunter, A. B. *A theory for record linkage*. Journal of the American Statistical Society 64:1183–1210.

[Gandon *et al.*, 2002] Gandon, F., Dieng, R., Corby, O., et Giboin, A. *Semantic Web and Multi-Agents Approach to Corporate Memory Management*. Dans 17th IFIP World Computer Congress. IIP Track-Intelligent Information Processing, Eds Musen M., Neumann B., Studer R., p. 103-115. August 25-30, 2002, Montreal.

[Gandon, 2002] Gandon, F. *Distributed Artificial Intelligence and Knowledge Management: ontologies and multi-agent systems for a corporate semantic web.* Scientific philosopher doctorate thesis in informatics, INRIA and University of Nice - Sophia Antipolis, November 2002.

[Gangemi *et al.*, 1999] Gangemi, A., Pisanelli, D.M., Steve, G. *An Overview of the ONIONS Project: Applying Ontologies to the Integration of Medical Terminologies*. Dans Data and Knowledge Engineering, 1999, vol.31, pp. 183-220, 1999.

[Garey et Johnson, 1979] Garey, M. R., Johnson, D. S. *Computers and Intractability: A Guide to the Theory of NPCompleteness.* Freeman & Co, New York, 1979.

[Garshol, 2003] Garshol, L. M. *Living with topic maps and RDF*. Dans proceedings of XML Europe 2003, 5-8 May 2003, organized by IDEAlliance, London, UK. http://www.ontopia.net/topicmaps/materials/ tmrdf.html

[Garshol, 2003b] Garshol, L. M. *The RTM RDF to topic maps mapping: Definition and introduction*. http://www.ontopia.net/topicmaps/materials/rdf2tm.html

[Ghidini et Giunchiglia, 2001] Ghidini, C. et Giunchiglia, F. *Local Models Semantics, or Contextual Reasoning = Locality + Compatibility*. Artificial Intelligence, 127(2):221-259. 2001.

[Giunchiglia et Serafini, 1994] Giunchiglia, F. et Serafini, L. *Multilanguage hierarchical logics, or: How we can do without modal logics*. Dans Artificial intelligence, Vol. 65, No. 1. (1994), pp. 29-70.

[Giunchiglia et Shvaiko, 2003] Giunchiglia, F. et Shvaiko, P. *Semantic matching.* Dans Proc. IJCAI 2003 Workshop on ontologies and distributed systems, Acapulco (MX), pages 139–146, 2003.

[Giunchiglia et Yatskevich, 2004] Giunchiglia F. et Yatskevich M. *Element Level Semantic Matching*. dans Meaning Coordination and Negotiation workshop at ISWC'04. Hiroshima, Japan, 2004.

[Giunchiglia *et al.*, 2004] Giunchiglia, F., Shvaiko, P. et Yatskevich, M. *S-Match: an algorithm and an implementation of semantic matching*. Dans Proceedings of ESWS 2004, Heraklion (GR), pages 61–75, 2004.

[Goldstein et Bobrow, 1980] Goldstein I. et Bobrow D. *Descriptions for a programming environment*. Dans Proc. of the first Annual conference on artificial intelligence, AAAI-1, Stanford Cal., August 18-21, 1980.

[Gómez-Pérez *et al.*, 2004] Gómez-Pérez, A., Fernández-López, M., Corcho, O. *Ontological Engineering*. XII, 403 p., 159 illus., ISBN: 1-85233-551-3. 2004.

[Grosof *et al.*, 2003] Grosof, B., Horrocks, I., Volz, R. et Decker, St. *Description Logic Programs: Combining Logic Programs with Description Logic*. In Proc. of WWW2003, May 2003.

[Gruber, 1993] Gruber, T.R. *A translation approach to portable ontologies*. Knowledge Acquisition, 5(2):199-220, 1993.

[Guarino et Giaretta, 1995] Guarino, N. et Giaretta, P. *Ontologies and knowledge bases- towards a terminalogical clarification.* Dans N.J. Mars, editor, Towards Very Large Knowledge Bases - Knowledge Building and Knowledge Sharing 1995, pages 25--32. IOS Press, Amsterdam, 1995.

[Hameed *et al.*, 2003] Hameed, A., Preece, A., et Sleeman, D. *Ontology Reconciliation*, Dans Staab, S and Studer, R, Eds. Handbook on Ontologies in Information Systems, pages pp. 231-250, 2003.

[Horrocks *et al.*, 2003] Horrocks, I. , Patel-Schneider, P. F. et van Harmelen, F. *From SHIQ and RDF to OWL: The making of a web ontology language*. Journal of Web Semantics, 1(1):7-26, 2003.

[Jaro, 1989] Jaro, M. A. *Advances in record-linkage methodology as applied to matching the 1985 census of Tampa*, Florida. Journal of the American Statistical Association 84:414–420.

[Jaro, 1995] Jaro, M. A. *Probabilistic linkage of large public health data files* (disc: P687-689). Statistics in Medicine 14:491–498.

[Kalfoglou et Schorlemmer, 2003a] Kalfoglou, Y. et Schorlemmer, M. *If-map: an ontology mapping method based on information flow theory.* Journal of data semantics, 1:98–127, 2003.

[Kalfoglou and Schorlemmer, 2003b] Yannis Kalfoglou, Y. et Schorlemmer, M. *Ontology mapping: the state of the art*. The Knowledge Engineering Review, 18(1):1–31, 2003.

[Klein, 2001] Klein, M. *Combining and relating ontologies: an analysis of problems and solutions*. Proc. of the IJCAI-01 Workshop on Ontologies and Information Sharing , Seattle, USA, August 4-5, 2001, p. 53-62.

[Levi, 1972] Levi, G., A Note on the Derivation of Maximal Common Subgraphs of Two Directed or Undirected Graphs. Calcolo, Vol. 9, pp. 341-354, 1972.

[Li et Clifton, 2000] Li, W.S. et Clifton, C. *Semint: a tool for identifying attribute correspondences in heterogeneous databases using neural networks*. Data Knowl.Eng., 33(1):49– 84, 2000.

[Lovin, 1968] Lovins, J.B. *Development of a stemming algorithm*. Mechanical Translation and Computational Linguistics, 11: 22-31, 1968.

[Mädche et Staab, 2002] Mädche, A. et Staab, S. *Measuring similarity between ontologies.* In Proc. Of the 13th Int. Conference on Knowledge Engineering and Management (EKAW-2002), Siguenza, Spain, October 2002. Springer-Verlag.

[Madhavan *et al.*, 2001] Madhavan, J., Bernstein, P. et Rahm, E. *Generic schema matching with cupid.* Dans Proceedings of the 27th International Conference on Very Large Data Bases, pages 49–58. Morgan Kaufmann Publishers Inc., 2001.

[Marino, 1993] Marino, O. *Raisonnement classificatoire dans une représentation multi-points de vue.* PhD thesis, Université Joseph Fourier – Grenoble 1. 1993.

[Martin, 1979] Martin, W. *Description and Specialization of Concepts*. Dans Patrick Winston and Richard Brown, Eds. Artificial Intelligence, MIT Press, Cambridge. 1979.

[Maynard et Ananiadou, 1999] Maynard, D.G. et Ananiadou, S. *Term extraction using a similarity-based approach*. In Recent Advances in Computational Terminology. John Benjamins, 1999.

[McCarthy, 1993] McCarthy, J. *Notes on formalizing context*. Dans Proc. of the 13th International Joint Conference on Artificial Intelligence, IJCAI'93. 1993.

[McGregor, 1982] McGregor, J.J., *Backtrack Search Algorithms and the Maximal Common Subgraph Problem*. Software Practice and Experience, Vol. 12, pp. 23-34, 1982.

[McGuinness *et al.*, 2000] McGuinness, D.L., Fikes, R., Rice, J. et Wilder, S. *An environment for merging and testing large ontologies.* Dans Proceeding of KR, pages 483–493, 2000.

[Melnik *et al.*, 2001] Melnik, S., Garcia-Molina, H., et Rahm, E. *Similarity Flooding: A Versatile Graph Matching Algorithm*. Extended Technical Report, http://dbpubs.stanford.edu/pub/ 2001- 25.

[Miller, 1995] Miller, A.G. *Wordnet: A lexical database for english.* Communications of the ACM, 38(11):39–41, 1995.

[Milo et Zohar, 1998] Milo, T., Zohar, S. *Using schema matching to simplify heterogeneous data translation.* In: Proc24th Int Conf On Very Large Data Bases, pp. 122–133, 1998.

[Mitra *et al.*, 2000] Mitra, P., Wiederhold, G. et Kersten, M. A graphoriented model for articulation of ontology interdependencies. In: Proc Extending DataBase Technologies, Lecture Notes in Computer Science, vol. 1777. Springer, Berlin Heidelberg NewYork, 2000, pp. 86–100.

[Monge et Elkan, 1996] Monge, A., et Elkan, C. *The field-matching problem: algorithm and applications*. In Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, 1996.

[Neches *et al.,* 1991] Neches, R., Fikes, R., Finin, T., Gruber, T., Patil, R., Senator, T. et Swartout, W.R.. *Enabling Technology for Knowledge Sharing*. Dans AI Magazine, pp. 36-56, 1991.

[Newell, 1982] Newell, A. *The Knowledge Level*. Artificial Intelligence. 18. Pp. 87-127. Jan, 1982.

[Nguyen *et al.*, 1992] Nguyen, G. T., Rieu, D. et Escamilla, J. *An Object Model for Engineering Design*. Dans Proc. of European Conference on Object-Oriented Programming. ECOOP'92. Lecture Notes in Computer Science vol. 615. Springer-Verlag, 1992.

[Noy et Musen, 2001] Noy, N. et Musen, M. *Anchor-PROMPT: Using non-local context for semantic matching.* Dans Proc. IJCAI 2001 workshop on ontology and information sharing, Seattle (WA US), pages 63–70, 2001.

[Paolo *et al.*, 2003] Paolo, B., Fausto G., Frank v.H., Luciano S. et Heiner S.: *C-OWL: Contextualizing Ontologies*. International Semantic Web Conference 2003: 164-179

[Palopoli *et al.*, 1998] Palopoli, L., Sacca, D. et Ursino, D. *Semi-automatic, semantic discovery of properties from database schemas.* In: Proc Int. Database Engineering and Applications Symp. (IDEAS), IEEE Comput, pp. 244–253, 1998.

[Patel-Schneider *et al.*, 2004] Patel-Schneider, P.F., Hayes, P. et Horrocks, I. *OWL Web Ontology Language Semantics and Abstract Syntax.* W3C Recommendation 10 February 2004. http://www.w3.org/TR/owl-semantics/, February 2004.

[Pepper *et al.*, 2006] Pepper, S., Vitali, F., Garshol, L.M., Gessa, N., Presutti. V. *A Survey of RDF/Topic Maps Interoperability Proposals*. W3C Working Group Note 10 February 2006. http://www.w3.org/TR/rdftm-survey/

[Pinto et Martins, 2001] Pinto, H.S. et Martins, J.P. *A Methodology for Ontology Integration*. In Proceedings of the First International Conference on Knowledge Capture , ACM Press, 2001.

[Porter, 1980] Porter, M.F. *An Algorithm for Suffix Stripping*, Program, 14(3): 130-137, 1980.

[Rahm et Bernstein, 2001] Rahm, E. et Bernstein, P. *A survey of approaches to automatic schema matching*. VLDB Journal, 10(4):334–350, 2001.

[Ribière , 1998] Ribière, M. *Using Viewpoints and CG for the Representation and Management of a corporate memory in Concurrent Engineering*. Proc. of the 6th Int. Conference on Conceptual Structures (ICCS'98), Montpellier, August 10-12, 1998, Springer-Verlag, LNAI 1453.

[Ribière, 1999] Ribière, M. *Représentation et gestion de multiples points de vue dans le formalisme des graphes conceptuels*. Thèse de doctorat, université de Nice Sophia-Antipolis, avril 1999.

[Ribière et Dieng, 1997] Ribière, M., Dieng, R. *Introduction of Viewpoints in Conceptual Graph Formalism*. ICCS 1997: 168-182. 1997.

[Ribière et Dieng, 2002] Ribière, M., Dieng, R. *A Viewpoint Model for Cooperative Building of an Ontology*. ICCS 2002: 220-234. 2002.

[Shvaiko et Euzenat, 2005] Shvaiko, P., Euzenat, J. *A Survey of Schema-based Matching Approaches*. Journal on Data Semantics, 2005.

[Sowa, 1984] Sowa, J. Conceptual Structures: Information Processing In Mind and Machine. Addison-Wesley, 1984.

[Steve *et al.*, 1997] Steve, G., Gangemi, A. et Pisanelli, D.M. *Integrating medical terminologies with ONIONS methodology*. Dans H. Kangassalo and J.P. Charrel, eds.. Information Modelling and Knowledge Bases VIII, IOS Press, Amsterdam, 1997.

[Stumme et Mädche, 2001a] Stumme, G. et Mädche, A. *FCA-merge: bottom-up merging of ontologies*. Dans Proc. 17th IJCAI, Seattle (WA US), pages 225–230, 2001.

[Valtchev, 1999] Valtchev, P. Construction automatique de taxonomies pour l'aide à la représentation de connaissances par objets. Thèse d'informatique, Université Grenoble 1, 1999.

[Valtchev et Euzenat, 1997] Valtchev, P et Euzenat, J. *Dissimilarity measure for collections of objects et values*. In P. Coen X. Liu et M. Berthold, editors, Proc. 2nd Symposium on Intelligent Data Analysis., volume 1280, pages 259–272, 1997.

[Visser *et al.*, 1997] Visser, P.R.S., Jones, D.M., Bench-Capon, T.J.M., et Shave, M.J.R. *Analysis of ontology mismatches*. Dans AAAI. Spring Symposium on Ontological Engineering, 1997.

[Wache *et al.*, 2001] Wache, H., Voegele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H. et Huebner, S. *Ontology-Based Integration of Information - A Survey of Existing Approaches*. Proc. of the IJCAI-01 Workshop on Ontologies and Information Sharing , Seattle, August 4-5, 2001.

[Winkler, 1999] Winkler, W. E. *The state of record linkage and current research problems*. Statistics of Income Division, Internal Revenue Service Publication R99/04. http://www.census.gov/srd/www/byname.html. 1999.

Annexe

Fichier de configuration pour l'application ASCO

```
#    // Input Ontologies

OntologyPath = /app/ontologies/

OntologyFileName1 = A.owl
OntologyFileName2 = B.owl

BenchmarkFileName = AB.bm

##############################
#    Parameters for ASCO1    #
##############################

#   weight parameters for ASCO1
#   sum of each group should be equal to 1.0
asco1.SimLin.wName = 0.6
asco1.SimLin.wLabels = 0.2
asco1.SimLin.wComment = 0.2

asco1.wPred = 0.4
asco1.wSucc = 0.4
asco1.wSibl = 0.2

asco1.wNeib = 0.5
asco1.wPath = 0.5

asco1.wLing = 0.9
asco1.wStru = 0.1


#   tLing : threshold for deciding if similar linguistically
#   tTotal : threshold for deciding if similar
asco1.tLing = 0.5
asco1.tTotal = 0.6



##############################
#    Parameters for ASCO2    #
##############################

#   sum of each group should be equal to 1.0
#   weight parameters for ASCO2
asco2.SimLin.wName = 0.2
asco2.SimLin.wLabels = 0.2
asco2.SimLin.wComment = 0.6

asco2.wID = 0.15
asco2.wType = 0.08
asco2.wLabel = 0.05
asco2.wComment = 0.1
asco2.wDomain = 0.026
asco2.wRange = 0.026
asco2.wSubClassOf = 0.05
asco2.wSubPropertyOf = 0.05
asco2.wEquivalentClass = 0.026
asco2.wEquivalentProperty = 0.026
asco2.wSameAs = 0.026
asco2.wComplementOf = 0.026
asco2.wInverseOf = 0.026
asco2.wDifferentFrom = 0.026
asco2.wDisjointWith = 0.026
asco2.wOnProperty = 0.026
asco2.wAllValuesFrom = 0.026
```

```
asco2.wSomeValuesFrom = 0.026
asco2.wHasValue = 0.026
asco2.wCardinality = 0.026
asco2.wMinCardinality = 0.026
asco2.wMaxCardinality = 0.026
asco2.wDistinctMembers = 0.026
asco2.wIntersectionOf = 0.026
asco2.wUnionOf = 0.026
asco2.wOneOf = 0.026

#  threshold for fixpoint calculation
asco2.tIteration = 100
asco2.tDeltaChange = 0.0000001

asco2.tTotal = 0.5


##############################
#    Parameters for ASCO3    #
##############################

#  weight parameters for ASCO3
#  sum of each group should be equal to 1.0
asco3.SimLin.wName = 0.6
asco3.SimLin.wLabels = 0.2
asco3.SimLin.wComment = 0.2

#  value for "jump" (saut)
asco3.nRadius_Super = 1
asco3.nRadius_Sub = 1
```

Ontologie MVP-OWL en OWL

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
     <!ENTITY rdf     "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
     <!ENTITY rdfs    "http://www.w3.org/2000/01/rdf-schema#" >
     <!ENTITY xsd     "http://www.w3.org/2001/XMLSchema#" >
     <!ENTITY owl     "http://www.w3.org/2002/07/owl#" >
     <!ENTITY vp      "http://www-sop.inria.fr/ACACIA/ontologies/mvpowl.owl#">
   ]>

<rdf:RDF
  xmlns       ="&vp;"
  xmlns:vp    ="&vp;"
  xmlns:owl   ="&owl;"
  xml:base    ="http://www.w3.org/2002/07/owl"
  xmlns:rdf   ="&rdf;"
  xmlns:rdfs  ="&rdfs;"
>

<owl:Ontology rdf:about="">
  <imports rdf:resource="http://www.w3.org/2000/01/rdf-schema"/>
  <imports rdf:resource="http://www.w3.org/2002/07/owl"/>
  <rdfs:comment>This file specifies in RDF Schema format the
    built-in classes and properties that together form the basis of
    the RDF/XML syntax of MVPOWL.
    We do not expect people to import this file
    explicitly into their ontology. People that do import this file
    should expect their ontology to be an MVPOWL ontology.
  </rdfs:comment>
  <owl:versionInfo>01 Jan 2006, revised $Date: 2006/01/01 $</owl:versionInfo>
</owl:Ontology>

<rdfs:Class rdf:ID="Viewpoint">
  <rdfs:label>Viewpoint</rdfs:label>
  <rdfs:subClassOf rdf:resource="&rdfs;Class"/>
</rdfs:Class>

<rdfs:Class rdf:ID="PerspectiveViewpoint">
  <rdfs:label>Perspective Viewpoint</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Viewpoint"/>
</rdfs:Class>

<rdfs:Class rdf:ID="OpinionViewpoint">
  <rdfs:label>Opinion Viewpoint</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Viewpoint"/>
</rdfs:Class>

<rdfs:Class rdf:ID="ClassWithViewpoint">
  <rdfs:label>Class with Viewpoint</rdfs:label>
  <rdfs:subClassOf rdf:resource="&owl;Class"/>
</rdfs:Class>

<owl:ObjectProperty rdf:ID="belongToViewpoint">
  <rdfs:label>belong to viewpoint</rdfs:label>
  <rdfs:domain rdf:resource="&owl;Thing"/>
  <rdfs:range rdf:resource="#Viewpoint"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="characterisedBy">
  <rdfs:label>characterised by</rdfs:label>
  <rdfs:domain rdf:resource="#Viewpoint"/>
  <rdfs:range rdf:resource="&rdf;Property"/>
</owl:ObjectProperty>
```

```
<owl:ObjectProperty rdf:ID="onClass">
  <rdfs:label>on class</rdfs:label>
  <rdfs:domain rdf:resource="#ClassWithViewpoint"/>
  <rdfs:range rdf:resource="&owl;Class"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="onViewpoint">
  <rdfs:label>on Viewpoint</rdfs:label>
  <rdfs:domain rdf:resource="#ClassWithViewpoint"/>
  <rdfs:range rdf:resource="#PerspectiveViewpoint"/>
</owl:ObjectProperty>

<rdf:Property rdf:ID="INCL">
  <rdfs:label>INCL</rdfs:label>
</rdf:Property>

<rdf:Property rdf:ID="EXCL">
  <rdfs:label>EXCL</rdfs:label>
</rdf:Property>

<rdf:Property rdf:ID="EQUIV">
  <rdfs:label>EQUIV</rdfs:label>
</rdf:Property>

<Viewpoint rdf:ID="GeneralViewpoint">
  <rdfs:label>General Viewpoint</rdfs:label>
</Viewpoint>

</rdf:RDF>
```

Résumé

Dans cette thèse, nous étudions les problèmes de l'hétérogénéité et du consensus dans un Web sémantique d'entreprise. Dans le Web sémantique, une extension du Web actuel, la sémantique des ressources est rendue explicite pour que les machines et les agents puissent les « comprendre » et les traiter automatiquement, afin de faciliter les tâches des utilisateurs finaux. Un Web sémantique d'entreprise est un tel web sémantique dédié à une entreprise, une organisation.

L'objectif de cette thèse est de permettre la construction et l'exploitation d'un tel Web sémantique, dans une organisation hétérogène comportant différentes sources de connaissances et différentes catégories d'utilisateurs, sans éliminer l'hétérogénéité mais en faisant cohabiter entre l'hétérogénéité et le consensus dans l'organisation tout entière.

Dans la première partie, nous approfondissons le problème de l'hétérogénéité des ontologies. L'ontologie est un des éléments fondamentaux dans le Web sémantique. Plusieurs ontologies différentes peuvent co-exister dans une organisation hétérogène. Pour faciliter l'échange des informations et des connaissances encodées dans différentes ontologies, nous étudions des algorithmes permettant d'aligner des ontologies déjà existantes. Les algorithmes proposés permettent de mettre en correspondance les ontologies représentées dans les langages RDF(S) et OWL recommandés par le W3C pour le Web sémantique. Ces algorithmes sont évalués grâce à des campagnes d'évaluation des outils d'alignement d'ontologies.

Dans la deuxième partie, nous nous intéressons au problème de la construction de nouvelles ontologies dans une organisation hétérogène mais en prenant en compte différents points de vue, différentes terminologies des personnes, des groupes voire des communautés diverses au sein de cette organisation. Une telle ontologie, appelée ontologie multi-points de vue, permet de faire cohabiter à la fois l'hétérogénéité et le consensus dans une organisation hétérogène. Nous proposons un modèle de représentation des connaissances multi-points de vue, appelé MVP, et un langage d'ontologie multi-points de vue, qui est une extension du langage d'ontologie OWL, appelé MVP-OWL, pour permettre de construire et d'exploiter des ontologies multi-points de vue dans un Web sémantique d'entreprise.

Mots-clés : Web sémantique, Web sémantique d'entreprise, ontologie, alignement d'ontologies, modèle multi-points de vue, ontologie multi-points de vue, RDF(S), OWL.

Abstract

In this thesis, we study problems of heterogeneity and consensus in a corporate semantic Web. In the semantic Web, an extension of the current Web, information is given well-defined meaning so that machines and agents can « understand » and process them automatically, in order to facilitate the end-user tasks. A corporate semantic Web is a semantic Web dedicated to an enterprise or an organization.

The objective of this thesis is to allow the construction and the exploitation of such semantic Web, in a heterogeneous organization comprising various sources of knowledge and various user categories, without eliminating heterogeneity but by making heterogeneity and consensus cohabit in the whole organization.

In the first part, we study thoroughly the ontology heterogeneity problem. Ontology is one of the most important fundamental elements in the semantic Web. Several different ontologies can coexist in a heterogeneous organization. To facilitate information exchange where information is encoded by various ontologies, we study algorithms allowing to align already existing ontologies. The proposed algorithms enable to align ontologies represented in different ontology languages recommended by W3C for the semantic Web: RDF(S) and OWL. These algorithms are evaluated through alignment tool evaluation campaigns.

In the second part, we are interested in the problem of building new ontologies in a heterogeneous organization by taking into account different viewpoints, different terminologies of various users, groups or even communities in the organization. Such ontology, called multi-viewpoint ontology, enables to have both heterogeneity and consensus co-exist in a heterogeneous organization. We propose a multi-viewpoint knowledge representation model, called MVP, and a multi-viewpoint ontology language, called MVP-OWL, which is an extension of the ontology language OWL, to allow the construction and the exploitation of the multi-viewpoint ontology in a corporate semantic Web.

Keywords : Semantic Web, corporate semantic Web, ontology, ontology alignment, model multi-viewpoint, multi-viewpoint ontology, RDF(S), OWL.

Thèse préparée à l'INRIA Sophia Antipolis dans le projet ACACIA