Ministère de l'Enseignement Supérieur et de la Recherche scientifique

BADJI MOKHTAR-ANNABA UNIVERSITY
UNIVERSITE BADJI MOKHTAR-ANNABA

جامعة باجي مختار - عنابة

Faculté des Sciences de l'Ingénieur
Département d'Informatique

Année : 2016/2017

THESE

Présentée en vue de l'obtention du diplôme de **DOCTORAT en Informatique**

Conception d'outils pour la fusion automatique d'ontologies

Option
Informatique

Par
Siham AMROUCH-BERROUK

Devant le jury

Président:	**Khadir Med. Tarek**	Pr. Université Badji Mokhtar - Annaba
Rapporteur:	**Mostefai Sihem**	MCA Université de Constantine 2- Constantine
Examinateur:	**Seridi Hassina**	Pr. Université Badji Mokhtar - Annaba
Examinateur:	**Lafifi Yacine**	Pr. Université 8 Mai 1945- Guelma
Examinateur:	**Belhadef Hacene**	MCA Université de Constantine 2- Constantine

Année Universitaire : 2016/2017

الملخص

تعرف البيانات على الويب الكلاسيكي بعدم تجانسها و حجمها المتزايد، مما يصعب عملية البحث على المعلومات. الويب الدلالي هو رؤية أخرى للويب الحالي، أين تم دمج دلالات البيانات لتسهيل عملية البحث على المعلومات. على الويب الدلالي، يتم تمثيل البيانات بشكل رئيسي عن طريق الأنطولوجيات التي تشكل قاموس مفردات مشترك (مع الحفاظ على غنى الدلالات) لتمثل ميدانا معينا .لذلك، فإن هناك عدة هيئات مختلفة تقوم بتطويرأنطولوجيات تمثل ميادين مشتركة أو متداخلة، مما يساهم في زيادة عدد الأنطولوجيات يوما بعد يوم. إن تطوير الأنطولوجيات عملية مكلفة و معقدة، لذلك غالبا ما يكون من الأنسب استخدام أنطولوجيات موجودة في ميدان معين بدلا من بناء أنطولوجيات جديدة. إذن، حتى نحقق تفاعلات و تعاملات بينية آلية و ذكية بين مختلف المستخدمين (إنسان أو آلة) و / أو التطبيقات غير المتجانسة، فان تقنية دمج الأنطولوجيات تعتبر حلا واعدا.

إن أنظمة دمج الأنطولوجيات الموجودة في أدب الأنطولوجيات تدرج التحليل اللغوي الذي يعتمد على معالجة اللغة الطبيعية لمطابقة المفاهيم و الخصائص و كذلك الأفراد. هذا من شأنه أن ينقص من النوعية الدلالية للمطابقات المحصل عليها لأنه (التحليل اللغوي) يعتمد على سياق محدد.

لمجابهة هدا النوع من الإشكال، نقترح خوارزمية جديدة لمطابقة و/أو دمج الأنطولوجيات أول-دل، بحيث تدرج هذه الخوارزمية تقنية مطابقة الأفراد باستعمال أشجار القرار، لتحسين مردودية تقنيات مطابقة المفاهيم و الخصائص المعتمدة على التحليل اللغوي. إن استعمال المصنف المعتمد على أشجار القرار هو ما يميز نظامنا، لأنه يتجنب نقائص تقنيات التحليل اللغوي. إن تقنيات المطابقة هذه، هي أيضا مدعمة بتقنية مطابقة أخرى تعتمد على التحليل الهيكلي للأنطولوجيات الأصلية و تستعمل قواعد استكشاف خاصة لاكتشاف المزيد من المطابقات. إن المطابقات المحصل عليها هي دلاليا مراجعة، منقاة و مجمعة قبل أن تستعمل لإنتاج الأنطولوجيا المدمجة الكلية.

يوفر نظامنا طريقة دمج آلية لا تتطلب أي تدخل بشري للحصول على أنطولوجيا كاملة، منسقة ومتماسكة انطلاقا من الأنطولوجيات الأصلية.

نظرا لكون إطار العمل المقترح يستعمل عدة أشكال من الدلالات الممثلة في الأنطولوجيات الأصلية و يدرج وحدة للمصادقة الآلية على المطابقات الأولية، فان الطريقة المقترحة تحسن دقة المطابقات المحصل عليها. إن خوارزميتنا هذه، تم تطبيقها على عدة حالات اختبار من شركة "مبادرة تقييم مطابقة الأنطولوجيات لعام 2015" و أعطت نتائج مشجعة و منافسة للأنظمة المصنفة عاليا.

الكلمات المفتاحية: دمج الأنطولوجيات، مطابقة الأفراد، أول-دل، التشابه المعجمي-الدلالي، وورد-نات، تقنية أشجار القرار، ويكا

Résumé

Les données du web sont connues par leur grande hétérogénéité et leur volume croissant ce qui ne facilite pas la recherche d'information. Le web sémantique est une autre vision du web à laquelle est intégrée la sémantique des données afin d'en faciliter la recherche. La sémantique des données est représentée principalement par des ontologies qui constituent un vocabulaire commun (et une même sémantique) couvrant un domaine particulier. Par conséquent, différents consortiums développent des ontologies dans les domaines communs ou qui se chevauchent, augmentant de jour en jour le nombre d'ontologies. La construction d'une ontologie étant une tâche ardue. Il est souvent plus indiqué d'utiliser des ontologies existantes d'un domaine plutôt que d'en construire de nouvelles. Alors, pour permettre une interopérabilité automatique et intelligente entre les utilisateurs humains et / ou applications hétérogènes, la fusion d'ontologies est une solution prometteuse.

Les systèmes d'alignement et/ou de fusion d'ontologies qui existent dans la littérature intègrent l'analyse linguistique basée sur le Traitement de Langue Naturelle (TLN) pour aligner les classes, les propriétés ainsi que les instances. Ceci peut réduire la qualité sémantique des mappings obtenus parce qu'elle (l'analyse linguistique) dépend d'un contexte bien spécifié. Pour s'attaquer à ce type de problème, nous proposons un nouveau algorithme d'alignement et / ou de fusion d'ontologies OWL-DL qui intègre une technique d'alignement des instances basée sur les arbres de décision, pour améliorer les performances des techniques d'alignement des classes et des propriétés, basées sur l'analyse linguistique. L'utilisation d'un classifieur basé sur les arbres de décision est ce que caractérise notre système, parce qu'il évite les limitations des techniques de TLN.

Ces techniques d'alignement sont également supportées par une technique d'alignement basée sur l'analyse structurelle des ontologies sources qui applique des heuristiques spécifiques pour détecter plus de mappings. Les mappings obtenus sont sémantiquement vérifiés, raffinés et combinés avant d'être utilisés pour générer l'ontologie fusionnée globale.

Notre système propose une approche automatique qui ne nécessite aucune intervention humaine pour obtenir une ontologie complète, cohérente et consistante à partir d'ontologies sources.

Vu que notre Framework exploite plusieurs formes de sémantiques représentées dans les ontologies sources et intègre un module de validation des mappings initiaux, notre approche améliore la précision des mappings obtenus. Notre algorithme est appliqué sur plusieurs cas de test de la compagnie OAEI'2015 et a donné des résultats encourageant et compétitifs avec les systèmes les mieux classés.

Mots clés : Fusion d'ontologies, Alignement des instances, OWL-DL, Similarité lexico-sémantique, WordNet, Arbres de décision, Weka.

Abstract

Web data are known by their great heterogeneity and increasing volume which does not facilitate the information retrieval. The semantic web is a different vision of the actual web to which is integrated the semantics of data to facilitate the research. The data semantic is mainly represented by ontologies which constitute a common vocabulary (with the same semantic) covering a particular area. Therefore, various consortiums develop ontologies that cover common or overlapping areas, which increase the number of ontologies from day to day. The construction of ontology is a hard task. It is often more appropriate to use existing ontologies rather than building new ones from scratch. So, to enable automatic and intelligent interoperability between human users and / or heterogeneous applications, the ontology merging is a promising solution.

Existing systems for ontology matching and /or ontology merging apply linguistic analysis based on Natural Language Processing (NLP) to match classes, properties and instances. This can reduce the semantic quality of matching results because it is context dependant. To address this issue, we propose a new algorithm for matching and / or merging OWL-DL ontologies that incorporates decision tree-based instance matching to boost the performance of linguistic-based matchers. Using a decision tree-based classifier is what characterizes our system, because it avoids the limitations of NLP techniques. These matchers are also enhanced by a structure-based matcher that applies specific heuristics to detect more mappings. Obtained mappings are semantically verified, refined and combined before being used to generate the global merged ontology.

Our system is based on an automatic approach that does not require any human intervention to get a complete, coherent and consistent merged ontology with regards to source ontologies.

Since our Framework exploits many forms of semantics represented in the sources ontologies and integrates a module to validate initial mappings, our algorithm improves the accuracy of the obtained mappings. Our algorithm is applied to various test cases of the Ontology Alignment Evaluation Initiative campaign, 2015 (OAEI'2015) and shows encouraging and concurrent results, with top ranked systems.

Keywords: Ontology merging, Instance matching, OWL-DL, Lexico-semantic similarity, WordNet, Decision trees, Weka.

Dédicaces

A ma chère mère,
La lumière de ma vie, qui n'a cessé de prier

pour que j'atteigne mes objectifs dans la vie.

A mon cher père,
La source et l'origine de tous mes succès.

A mon mari,
Mon compagnon de route qui
n'a cessé de m'aider et de m'encourager
pour que j'arrive au terme de ce travail.

A ma fille, Takwa Allah
le puits d'amour dans lequel
je puise chaque jour,
qu'elle trouve ici l'expression de mon profond amour.

A tous ceux qui me sont chers…
Je dédie les fruits de ce modeste travail

Siham

Remerciements

Louanges à Allah, qui m'a donné la force, la patience et la volonté d'arriver au terme de ce travail.

Je tiens à exprimer toute ma reconnaissance et mon profond respect à mon encadreur, Dr. Sihem MOSTEFAI pour son encadrement, sa confiance et son soutien. Son soutien et ses précieux conseils et recommandations m'ont permis de mener ce travail dans de très bonnes conditions. Merci beaucoup Dr. Sihem MOSTEFAI.

Je voudrais remercier Prof. Med Tarek KHADIR de m'avoir fait l'honneur de présider mon jury. Je veux également exprimer toute ma gratitude à Pr. Hassina SERIDI, Pr. Yacine LAFIFI et Dr. Hacene BELHADEF d'avoir accepté de rapporter mon manuscrit de thèse.

J'adresse également mes plus sincères remerciements au Prof. Mohammed BOURAS de m'avoir accordé avec beaucoup de gentillesse, de faire un stage au sein du laboratoire d'informatique CERRAL à l'université Lumière, Lyon2.
Je souhaite exprimer toute ma gratitude au Dr. Muhammad FAHAD, chercheur au laboratoire CERRAL, pour l'opportunité qu'il m'a offerte toute au long de la période de stage au labo, ainsi que son collaboration et son aide à distance après le stage. Tout au long de ce stage, j'ai pu bénéficier de ses conseils et j'ai pu apprécier sa disponibilité.

Je voudrais aussi profiter de cette occasion pour remercier tous ceux qui ont contribué de près ou de loin à l'élaboration de ce travail, en particulier ma mère, mon père et mon mari.

Liste des tables

Liste des figures

Sommaire

1. Introduction générale.

Depuis son apparition à l'aube des années 90, le web classique a été conçu comme étant un moyen pour faciliter la propagation et la diffusion d'informations et de données entre les différents utilisateurs et/ou applications éloignés. Il a permis, avec succès, aux utilisateurs, la récupération de données ainsi que l'échange d'informations entre compagnies différentes. Cependant, de plus en plus de sites Web voient le jour, et la quantité de données et de pages web a grandi, considérablement (plus de 3,1 milliards de pages Web, actuellement). En conséquence, la navigation et la recherche de données sur le Web sont devenues problématiques. La principale raison pour une telle situation est que le web lui-même a été vu, par des machines, comme étant un ensemble de sites web accessibles à travers des hyperliens, ce qui ne fournit aucune indication spécifique pour ces machines à propos de la signification ce site web ou dans quel but il a été conçu. Alors, pour permettre aux ordinateurs de « comprendre » le contenu des pages Web, et donc de fournir de meilleurs résultats de recherche, il est devenu nécessaire d'ajouter des informations sémantiques aux informations structurelles composant le web actuel. Le web sémantique est devenu alors une nécessité pour atteindre ces objectifs.

Dans ce qui suit, nous allons introduire notre travail de thèse qui s'articule autour du problème de la fusion automatique des ontologies hétérogènes, tout en spécifiant le contexte ainsi que les motivations de recherche. Ensuite, nous allons brièvement décrire le problème de la fusion d'ontologies et nous terminons par la présentation de l'organisation de la thèse.

1.1. Contexte et motivations:

Les ontologies représentent la « colonne vertébrale » du web sémantique vu qu'elles véhiculent la sémantique (ou la signification) formelle des ressources web d'un domaine spécifique. En effet, elles sont considérées comme un modèle standard pour l'organisation et la représentation des connaissances, cela permet le partage de données et assure une meilleure interopérabilité entre utilisateurs et/ou applications distants, car elles fournissent des représentations formelles des connaissances décrivant le domaine d'application, ce qui peut aider les ordinateurs à comprendre le contenu des pages web.

La découverte et le développement automatique ou semi-automatique d'ontologies est le seul moyen pratique pour transiter vers le web sémantique. La fusion d'ontologies est une des stratégies permettant le développement de nouvelles ontologies à partir des ontologies préexistantes. (Richardson et Mazlack, 2004). Pour modéliser des problèmes et des domaines, les ontologies fournissent une bibliothèque de classes et de relations qui sont faciles à réutiliser (Genesereth et al., 1992), (Uschold et Gruninger, 1996), (Maedche et al., 2003). Elles peuvent être utilisées à plusieurs fins, entre autres: Pour faciliter la communication entre utilisateurs hommes et / ou machines, Pour assurer l'interopérabilité et la communication entre systèmes ou logiciels distants, ainsi que pour améliorer la conception et la qualité des systèmes (Waralak et Siricharoen, 2007). Les ontologies décrivent une spécification explicite d'une conceptualisation, qui est utilisée pour aider les programmes et les utilisateurs humains à partager les connaissances (Gruber, 1993). D'un point de vue de la technologie, une ontologie peut être considérée comme étant un dépôt de classes (concepts) ressemblant à une base de données qui représente un dépôt de données (Waralak et Siricharoen, 2007). Elle se compose d'un ensemble de classes qui sont décrites par un ensemble de propriétés et liées les unes aux autres par un ensemble de relations sémantiques. En outre, les classes sont organisées de façon hiérarchique pour montrer lesquelles de ces classes sont plus générales ou plus spécifiques en décrivant un contexte donné, ce qui résulte en une hiérarchie de classes.

Alors, pour supporter l'interopérabilité sémantique, différents consortiums développent des ontologies dans des domaines qui couvrent différents aspects et / ou qui se chevauchent, ayant pour conséquence l'explosion du nombre d'ontologies disponibles sur le web. Alors, pour permettre une interopérabilité automatique et intelligente entre les utilisateurs humains et / ou applications hétérogènes, la fusion de ces ontologies est une solution prometteuse. Construire des ontologies de domaine est une tâche complexe et coûteuse qui consomme beaucoup de temps et de ressources machines. Mais, par la fusion des ontologies individuelles qui représentent des fragments de connaissances, on peut minimiser les coûts de création de ces ontologies. Il est également crucial de fusionner les ontologies pour la construction d'autres ontologies pour les domaines évolutionnaires, interdisciplinaires et spécifiques. En

effet, les ontologies des domaines évolutionnaires subissent des changements à chaque fois que les domaines évoluent. La fusion de plusieurs versions des ontologies de base produit de nouvelles ontologies plus riches et plus complètes. En plus, on peut créer des ontologies interdisciplinaires par la fusion des sous ontologies spécifiques aux sous domaines. On peut également, avoir besoin de fusionner les ontologies spécifiques aux domaines avec des ontologies plus générales et de nouvelles informations peuvent avoir besoin d'être fusionnées avec des ontologies existantes. Par conséquent, la fusion d'ontologies est un vrai challenge face à la transition vers le web sémantique.

1.2. Description du problème de la fusion d'ontologies :

La fusion d'ontologies est le processus de création d'une nouvelle et unique ontologie cohérente à partir de deux (ou plus) ontologies existantes et liées au même domaine (Ghidini et Giunchiglia, 2003). Le processus de fusion a permis de mettre en évidence différentes formes d'hétérogénéité entre les ontologies sources dans différents domaines d'applications tels que les systèmes multi-agents, la gestion du cycle de vie du produit, la maintenance assistée par ordinateur, le E-Learning, les services Web et le Web sémantique. Nous estimons que le processus de création d'une nouvelle ontologie par la fusion de deux (ou plus) ontologies source est plus facile que de créer une nouvelle ontologie à partir de zéro.

Dans la littérature, il existe deux grandes catégories d'approches de fusion d'ontologies qui sont principalement basées sur l'alignement d'ontologies, la fusion complète et l'ontologie bridge (Bruijn et al., 2006) , voir figure 1:

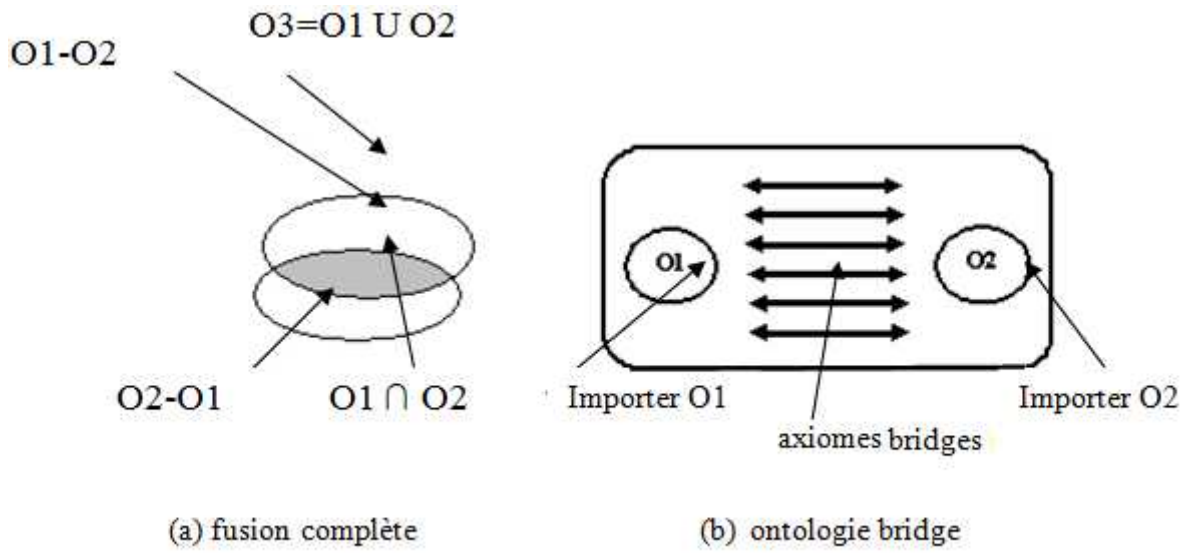


Figure 1. Les approches de fusion d'ontologies.

Dans la première approche, le processus de fusion donne une seule ontologie de sortie qui contient les ontologies sources. Les exemples de cette approche sont *Prompt* (Noy et Musen, 2000), *Chimaera* (McGuinness et al., 2000), etc. Dans la seconde approche, le processus de fusion donne une ontologie bridge qui importe les ontologies sources et leur associe des axiomes bridges ou des règles d'articulation représentant les mappings entre les classes des ontologies sources. Les exemples de cette approche sont *OntoMerge* (Dou, Mcdermott et Qi, 2002), *ONION* (Mitra et Wiederhold, 2002), etc.

Dans la littérature également, les approches de fusion d'ontologies peuvent être symétriques ou asymétriques (Raunich et Rahm, 2012). Les anciens travaux sur la fusion d'ontologies (tels que *Prompt*, *Chimaera*, *OntoMerge*, etc.,) exploitent une solution symétrique. Ils traitent toutes les ontologies sources avec le même niveau de priorité lors de la fusion et l'ontologie fusionnée peut (ou non) ressembler à l'une des ontologies sources. Alors que, les travaux récents tels que *OM* (Guzmán-Arenas et Cuevas, 2010) et *Atom* (Raunich et Rahm, 2011) exploitent une approche asymétrique. Ici, le processus de fusion donne la priorité à une des ontologies sources et l'autre se fusionne dans l'ontologie prioritaire qui maintient sa structure. Un exemple de solution asymétrique est représenté par la figure 2, où la structure de l'ontologie prioritaire O_1 est conservée dans l'ontologie fusionnée O_3. Par exemple, la classe D dans O_1 est

maintenue en tant que sous-classe de la classe B dans l'ontologie O_3, indépendamment de son conflit structurel avec O_2 où elle est une sous-classe de E.

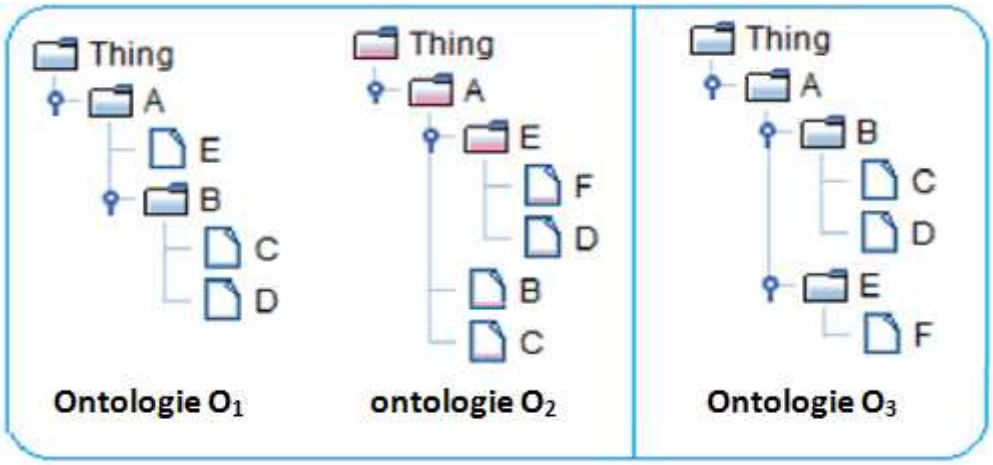


Figure 2. Exemple de la fusion asymétrique.

Les techniques de fusion citées ci-dessus peuvent être appliquées selon les exigences du processus de fusion. Il a été convenu par les chercheurs que pour un processus de fusion, il n'y a pas de solution unique (Raunich et Rahm, 2012). Mais, tout dépend des exigences du processus de fusion et où et quand la fusion est exécutée.

Les systèmes actuels d'alignement et / ou de fusion d'ontologies intègrent des modules d'alignement des instances pour améliorer les performances des modules d'alignement des classes et des propriétés qui sont basés sur l'analyse linguistique des termes ontologiques. D'après la littérature, les modules d'alignement des instances sont également basés sur le calcule de similarité lexicale et / ou sémantique entre tous (ou un sous ensemble) les pairs d'instances appartenant aux ontologies sources. Ceci, ne conduit pas seulement à la consommation d'énormément de ressources machines (temps d'exécution et espace mémoire) parce que l'espace de recherche est très vaste, mais en plus, il génère des mappings de mauvaise qualité, car il n'évite pas les limitations des techniques linguistiques basés sur le Traitement de Langue Naturelle (TLN).

Pour s'adresser à ces problèmes, le module d'alignement des instances intégré dans notre système utilise la technique des arbres de décision pour classifier les instances de l'ontologie cible avec leurs instances similaires de l'ontologie source sous la classe similaire correspondante (deux classes sont jugées similaires si elles partagent suffisamment d'instances). L'exécution automatique et parallèle de la classification

par le classifieur basé sur les arbres de décision ne réduit pas seulement l'espace mémoire et le temps d'exécution écoulé, mais évite également les limitations de Traitement de Langue Naturelle car ils (les arbres de décision) sont indépendants de tous type de dictionnaire et / ou de base de données lexicale.

En plus, vu que la plupart des systèmes existant sont basés sur l'alignement des noms des entités ontologiques, ceci peut rater énormément de mappings intéressants, tels que les mappings entre les propriétés d'objets suivants : (*assignedTo* Vs *hasReviewer*), (*hasBeenAssignedTo* Vs *reviewerOfPaper*), (*writtenBy* Vs *reviewWrittenBy*), etc.

Pour combler ces lacunes, notre système intègre également, un module d'alignement basé sur l'analyse structurelle des ontologies sources qui diffuse l'ensemble des mappings obtenus par le module d'alignement au niveau entité (combinaison des mappings obtenus par l'alignement au niveau schéma et ceux obtenus au niveau extensionnel) à travers les structures des ontologies sources (pour obtenir plus de mappings) en utilisant certaines heuristiques spécifiques.

Les mappings obtenus sont sémantiquement vérifiés, raffinés et combinés avant d'être utilisés pour générer l'ontologie fusionnée globale.

Notre algorithme de Fusion Automatique d'Ontologie qui est basée sur un algorithme de Mapping d'Ontologies Flexible est nommé, AOM-FOM (Automatic Ontology Merging based on a Flexible Ontology Matcher). Le caractère flexible de notre algorithme est assuré par le fait qu'on peut activer le /les module d'alignement intégré dans notre système selon le contenu et la structure des ontologies sources. Par exemple, si les instances de classes ne sont pas représentées dans l'une (ou les deux) des deux ontologies sources (pour des raisons de sécurité, par exemple), on peut désactiver le module d'alignement au niveau extensionnel. D'un autre côté, si l'une (ou les deux) des deux ontologies sources ne contiennent pas d'informations linguistiques représentées (tels que les noms de classes et / ou de propriétés), on peut désactiver le module d'alignement au niveau schéma et retenir les mappings au niveau extensionnel.

Par ailleurs, nous allons, également baser notre algorithme sur une stratégie de priorité qui donne la priorité à la recherche des mappings candidats à la fusion dans les partitions disjointes. Cette stratégie minimise considérablement le nombre de comparaisons pour détecter les mappings candidats à la fusion, et donc augmente les

performances du système et réduit la consommation des ressources machines, i.e. le temps d'exécution et la mémoire de stockage.

L'algorithme proposé reçoit en entrée deux ontologies OWL-DL et donne en sortie une seule ontologie OWL-DL qui est plus complète et cohérente. Il se compose de trois phases principales. Après avoir importé les deux ontologies sources en format OWL-DL, la première phase consiste à prétraiter les ontologies sources pour préparer et faciliter l'étape d'extraction des classes candidates à la fusion. Les résultats de cette phase sont les graphes OWL-DL représentant les ontologies sources et l'ensemble des termes ontologiques sous leurs formes de base. La deuxième phase consiste à établir un processus d'alignement entre les ontologies sources et extraire la liste de mappings initiaux entre leurs classes et propriétés. Ensuite, elle s'occupe de valider ces mappings et extraire la liste de mappings finaux. Le résultat de cette phase est représenté par deux ensembles: Un ensemble de classes et de propriétés similaires et un ensemble de classes et de propriétés différentes.

La dernière phase consiste à fusionner les classes ainsi que leurs propriétés similaires et copier celles non liées. Le résultat de cette étape est l'ontologie fusionnée résultante.

1.3. Organisation de la thèse

En plus de l'introduction générale qui introduit notre travail de recherche, en décrivant le contexte d'utilisation et d'exploitation des techniques proposées et développées, suivi par les motivations nous encourageant à emprunter cette voie de recherche ainsi que la description de la problématique de la fusion d'ontologie, et la conclusion générale et l'ensemble de perspectives qui clôturent le manuscrit, nous avons structuré le corps de notre thèse en deux parties.

La première partie, présentant un état de l'art sur les techniques étudiées, exploitées et développées dans ce travail de thèse, est composée de trois chapitres.

Le premier chapitre intitulé, Concepts généraux sur les ontologies, fournit un survol sur la littérature du concept ontologique, ses principes de construction, ses langages de représentation et ses éditeurs de construction tout en présentant une classification des erreurs et anomalies de conception et de construction d'ontologies ainsi que

l'explication du problème d'hétérogénéité lors de l'évolution et la réutilisation d'ontologies.

Le deuxième chapitre intitulé, Fusion d'ontologies –État de l'art, présente un état de l'art sur la technique de la fusion d'ontologies. Nous détaillons alors les étapes du processus de fusion d'ontologies, ainsi que les différents types de problèmes à résoudre avant la fusion. A ce stade, le processus de découverte de correspondances, ses architectures, l'effet de l'aspect sémantique sur ses performances ainsi que ses différentes techniques sont détaillés. A la fin, ce chapitre présente un récapitulatif de quelques applications de la technique de la fusion d'ontologies suivi par un survol sur les principaux travaux de fusion d'ontologies existant dans la littérature.

Le troisième chapitre intitulé, Arbres de décision, décrit la technique de classification par arbres de décision que nous avons utilisé pour la détection des classes similaires entre les ontologies sources. Ce chapitre présente alors, les concepts de base concernant les arbres de décisions, leurs différentes formes de représentations, le processus d'apprentissage pour la construction du classifieur basé sur les arbres de décisions ainsi que le processus de reconnaissance pour classifier les classes d'une ontologies avec leurs classes similaires dans une autre ontologie, en utilisant ce classifieur. Enfin, ce chapitre présente les différents algorithmes de constructions des arbres de décision existant dans la littérature.

La deuxième partie présentant nos contributions, implémentations, évaluations et tests, pour la fusion des ontologies hétérogènes, est composée de deux chapitres :

Le quatrième chapitre intitulé, Conception du Framework AOM-FOM, présente la conception de l'algorithme de notre système AOM-FOM qui est un outil de fusion automatique d'ontologies qui utilise un algorithme flexible de mapping tout en détaillant les stratégies utilisées pour la détection des classes similaires. Nous montrons alors les différents modules d'alignement intégrés dans notre système qui sont le module d'alignement au niveau entité (qui combine les mappings obtenus par le module d'alignement au niveau schéma et ceux obtenus par le module d'alignement au niveau extensionnel) et le module d'alignement au niveau structurel, tout en décrivant comment nous avons utilisé la technique basée sur l'apprentissage

automatique des instances en utilisant des arbres de décision et comment nous les avons appliqués.

Le cinquième chapitre intitulé, Implémentation du système AOM-FOM, démontre l'implémentation de notre système de fusion automatique d'ontologies basé sur l'alignement flexible d'ontologies sources. Ensuite, il présente les résultats expérimentaux de notre système tout en évaluant ses différents approches et algorithmes, tout en comparant les résultats obtenus par notre système avec ceux obtenus par les systèmes de mappings et de fusion qui ont participé dans le Framework « Ontology Alignement Evaluation Initiative », (OAEI 2015). Enfin, différentes expérimentations pour fusionner des ontologies avec différents scénarios sont présentées.

Chapitre 1 . Concepts généraux sur les ontologies.

1. Introduction

Lors de la dernière décennie, la quantité d'informations et de connaissances disponibles et produites sur le web a augmenté d'une manière considérable. Dans ce sens, le problème majeur qui se pose est l'efficacité de la recherche et de la réutilisation de ces sources d'informations et/ou de connaissances. Selon (Kobayashi & Takeda, 2000), 85% des utilisateurs du web utilisent des moteurs de recherche pour trouver des informations. Dans (Voorhees, 1994), les auteurs affirment que malgré le succès des moteurs de recherche, les utilisateurs se trouvent fréquemment en face d'un certain nombre de problèmes tels que la précision et le rappel bas, des documents non-personnels ou hors date, des rapports multi-langues et multimédia insuffisants, etc. La plupart de ces problèmes sont liés à l'hétérogénéité des sources du web, en plus du manque de relations formelles entre les différents domaines de connaissances. La solution prometteuse qui est proposée par Tim Berners Lee consiste alors à reformuler le web actuel pour obtenir le fameux web sémantique. Tim Berners Lee déclare que « le web sémantique n'est pas un web séparé mais c'est une extension du web actuel, dans lequel une information est définie par un sens ou une sémantique bien déterminée », (Hendler, Lassila, & Berners-Lee, 2001). Ce sens et cette sémantique sont représentés par un concept fondamental qui forme le noyau du web sémantique : l'ontologie. Dans ce sens les ontologies ont joué un rôle technologique clé pour représenter et stocker les informations à travers des modèles qui préservent les sémantiques adressés par les domaines des applications porteuses. La définition la plus connue et la plus succincte d'une ontologie est celle de Thomson Gruber en 1993 (Gruber, 1993), qui définit une ontologie comme étant une spécification explicite d'une conceptualisation. Elle est structurée sous forme de classes hiérarchisées et de relations entre classes. Les ontologies sont alors largement utilisées pour résoudre les problèmes d'hétérogénéités sur le web vu leur capacité de fournir une signification explicite d'une information. Elle joue le rôle d'un vocabulaire commun pour les chercheurs qui ont besoin des connaissances formelles dans un domaine donné

favorisant ainsi le partage et l'interopérabilité des informations et/ou connaissances qu'elles modélisent.

Dans un environnement distribué, les ontologies établissent un vocabulaire commun pour les chercheurs d'une communauté pour inter-relier, combiner et communiquer les connaissances formulées à travers les expertises et les interactions en plus des liaisons des traitements de création, importation, capture, recherche et utilisation d'autres connaissances (Kotis, Vouros, & Stergiou, 2006).

Selon (Aubry, 2007), les ontologies permettent une formalisation informatique des connaissances. Cette représentation peut être cognitivement sémantique (ontologie destinée à être exploitée par l'utilisateur) ou automatiquement sémantique (ontologie destinée à être exploitée par la machine) ou une combinaison des deux. Dans ce travail, nous considérons une ontologie comme étant un outil pour la gestion des connaissances. Le but de ce chapitre est de présenter le concept d'ontologie à travers un survol sur la littérature du domaine. D'abord, nous présentons la notion d'ontologie, ses composants, ses domaines d'application et ses critères de classification. Puis, nous abordons le processus d'ingénierie ontologique, ses principes de base, ses méthodologies de construction, son cycle de vie et son processus de construction. Ensuite, nous présentons une classification des erreurs de construction d'ontologies suivie par un survol sur les différents langages de représentation et éditeurs de construction qui existent dans la littérature. Enfin, nous introduisons la notion de la réutilisation d'ontologies et nous terminons par une conclusion.

2. Notion d'ontologie

Pendant la dernière décennie, nous avons constaté une inspiration très intelligente d'une notion philosophique très ancienne vers le domaine d'informatique et d'intelligence Artificielle. Cette notion a rapidement constitué une voie de recherche très importante dans plusieurs communautés et surtout celles basées sur l'ingénierie et la gestion des connaissances ainsi que la réutilisation et l'ingénierie des systèmes. Il s'agit bien de la notion d'ontologie.

En informatique et en particulier en Intelligence Artificielle et Sciences cognitives, la notion d'ontologie est apparue pour la première fois vers les années 90,

où John McCarthy a fait une correspondance entre les concepts de base de l'Ontologie (philosophique) et l'activité de construire des théories logiques des systèmes de l'IA (Aubry, 2007). Puis, plusieurs acceptions et points de vue ont été proposés par les différents auteurs du domaine. Nous les présentons dans un ordre chronologique dans la section suivante:

D'abord, Neeches et al. ont défini l'ontologie en 1991 par: *« An ontology defines the basic terms and relations to define extensions of the vocabulary ».* i.e. *« Une ontologie définit les termes de base et les relations pour définir des extensions d'un vocabulaire ».* (Neeches et al., 1991). En 1993, Thomson Gruber a suggéré la définition la plus succincte dans la littérature: *« An ontology is an explicit specification of a conceptualization »,* i.e. *«Une ontologie est une spécification explicite d'une conceptualisation ».* (Gruber, 1993). Puis, Guarino l'avait définie en 1995 par: «*Une ontologie est une théorie logique qui permet une spécification explicite et partielle d'une conceptualisation»,* (Guarino, 1995). Ensuite, l'ontologie a été connue selon Borst en 1997 comme étant: *« An ontology is a formal specification of a shared conceptualization ». i.e. « Une ontologie est une spécification formelle d'une conceptualisation partagée »,* (Borst, 1997), et lors de cette même période elle a été définie par Swartout comme étant: *"une ontologie est un ensemble de termes structurés de façon hiérarchique, conçu afin de décrire un domaine et qui peut servir de charpente à une base de connaissances »,* (Swartout et al., 1997). Alors qu'en 1998, elle a été décrie par Studer à travers la suggestion: "*Une ontologie peut prendre différentes formes, mais elle inclura nécessairement un vocabulaire de termes et une spécification de leur signification. Cette dernière inclut des définitions et une indication de la façon dont les classes sont reliées entre elles* ». (Studer et al., 1998). Après cela, Gomez-Perez l'avait définie en 1999 par: *« Une ontologie apporte les moyens pour décrire explicitement la conceptualisation sous-jacente aux connaissances représentées dans une base de connaissances ».* (Gomez-Perez, 1999a). En plus, Brodeur a suggéré en 2004 qu' *« Elle peut prendre la forme d'un thésaurus, réseau sémantique, taxonomie, modèle conceptuel, répertoire de données, etc. »,* (Brodeur, 2004). Enfin, une synthèse de ces définitions donnée par Amrouch et Khadir, en 2009 déclare qu': « *Une ontologie est une structure formelle, qui peut être*

graphique, où les nœuds représentent les classes, et les arcs représentent les relations, modélisant ainsi explicitement les connaissances d'un domaine particulier de manière à faciliter la recherche, l'extraction, l'intégration et le partage d'informations entre les différents systèmes ou individus au sein de ce domaine ». (Amrouch et Khadir, 2009).

Bien que ces définitions offrent des points de vue différents mais complémentaires sur le concept ontologique, elles focalisent sur des critères communs qui représentent les caractères de base d'une ontologie, et qui sont:

- ✓ Formelle: Signifie que l'ontologie est exprimable à l'aide d'une logique pouvant être traitée par une machine (machinable).
- ✓ Spécification explicite: Signifie que les classes, relations, fonctions, instances et axiomes d'une ontologie, sont définis d'une manière déclarative.
- ✓ Partagée: Signifie que l'ontologie modélise les connaissances partagées entre les différents individus d'une communauté d'un domaine particulier.

3. Composants d'une ontologie

Tels que tous formalismes de représentation de connaissances, les ontologies se composent d'un certain nombre d'unités constructives pour décrire et représenter les connaissances de manière formelle. Ces unités ontologiques constructives sont spécifiées par Gomez-Perez dans (Gómez-Pérez, 1999a), basé sur la définition de Gruber en 1993. Elles sont décrites par: des classes, des relations, des fonctions, des axiomes et des instances.

3.1. Classes

Elles sont également connues comme étant les concepts de l'ontologie. Elles représentent des objets physiques et/ou logiques sous une forme hiérarchique. Les classes d'une ontologie sont extraites après la conceptualisation du domaine ciblé en fonction des objectifs à atteindre et de l'application qui va utiliser l'ontologie.

« A concept (a class) can be anything about which something is said and, therefore, could also be the description of a task, function, action, strategy, reasoning process, etc.» i.e. «*Un concept (une classe) est n'importe quelle chose au sujet de laquelle on peut dire quelque chose et, ainsi pourrait aussi être la description d'une tâche, d'une*

fonction, d'une action, d'une stratégie, d'un raisonnement, etc. ». (Gómez-Pérez, 1999a).

3.2. Relations

« Relations represent a type of interaction between classes of the domain. They are formally defined as any subset of a product of n sets, that is, $R : C_1 \times C_2 \times \ldots\ldots\ldots \times C_n$ » i.e. *"Les relations représentent un type d'interaction entre les classes d'un domaine. Elles sont formellement définies comme étant un sous ensemble du produit de n ensemble, c-à-d,* $R : C_1 \times C_2 \times \ldots\ldots\ldots \times C_n$ *"*.(Gómez-Pérez, 1999a).

Les relations représentent alors, une grande partie de la sémantique de l'ontologie à travers les étiquetages (Nom de relation) attribués à ces associations tels que la subsomption (spécification ou généralisation, *is-a*), l'instanciation (*instance-of*), la composition (*part-of*), l'affectation ou l'attribution (*associated-to*), etc.

3.3. Fonctions

« Functions are special case of relations in which the n-th element of the relationship is unique for the n-1 preceding elements, formally, functions are defined as :

$$\begin{array}{l} F : C_1 \times C_2 \times \ldots\ldots\ldots \times C_{n-1} \to C_n \\ E_n \mapsto f(E_1, E_2, \ldots\ldots\ldots E_{n-1}) \end{array}$$

». i.e.

Une fonction est un cas particulier d'une relation, où le $n^{ième}$ *élément de la relation est unique pour les n-1 éléments précédents et formellement une fonction est définie comme suit:*

$$\begin{array}{l} F : C_1 \times C_2 \times \ldots\ldots\ldots \times C_{n-1} \to C_n \\ E_n \mapsto f(E_1, E_2, \ldots\ldots\ldots E_{n-1}) \end{array}$$

(Gómez-Pérez, 1999a)

3.4. Axiomes

« Axioms are used to model sentenses that are always true » i.e. *Les axiomes sont utilisés pour modéliser des assertions qui sont toujours vraies.* (Gómez-Pérez, 1999a).

3.5. Instances

« Instances are used to represent elements ». i.e. Les instances sont utilisées pour représenter des éléments individuels. (Gómez-Pérez, 1999a)

4. Domaines d'applications

4.1. Ontologies dans les systèmes d'informations

Les ontologies sont développées et utilisées pour faciliter, la mise en œuvre et l'exploitation de certaines techniques et mécanismes nécessaires pour la gestion des connaissances et l'échange d'informations dans les différents systèmes d'informations et/ou applications logicielles. Dans (Uschold & Gruninger, 1996), les auteurs affirment que les ontologies sont développées pour offrir une aide dans au moins trois domaines d'application à savoir la communication entre les êtres humains, l'interopérabilité entre les Systèmes d'Informations hétérogènes, ainsi que la réutilisabilité et le partage d'informations. Nous pouvons donc résumer le rôle des ontologies dans le domaine des systèmes d'informations comme suit:

õ **La communication** : Une ontologie est un modèle standard qui décrit et spécifie explicitement son domaine, constituant ainsi un espace partageable entre les (sous) systèmes et/ou individus, où ils partagent leurs points de vue et compréhensions perspectives, éliminant ainsi tout risque de confusion ou d'incompréhension, et favorisant tout effort de collaboration et de partenariat.

L'interopérabilité entre les systèmes d'informations hétérogènes: Pour assurer une meilleure interopérabilité entre des systèmes d'informations hétérogènes, une ontologie permet de modéliser les connaissances (les classes ainsi que les règles qui les régissent) qu'ils (les systèmes d'informations) doivent partager. En d'autres termes, en jouant le rôle d'un format d'échange, où elle répertorie toutes les classes qui doivent être échangées par les applications, l'ontologie facilite la communication, la collaboration ainsi que la coopération entre différent systèmes d'informations. Cette coopération est principalement, dépendante de la flexibilité de communication entre les différents systèmes, car leurs actions et réactions résultent des messages envoyés et reçus.

õ **Soutenir la spécification et la conceptualisation de systèmes:** Une ontologie est un concept réutilisable et/ou partageable par plusieurs (sous) systèmes, ce qui permet de faciliter l'acquisition d'informations, l'analyse de données ainsi que la spécification des besoins. En plus, sa structuration plus lisible et compréhensible permet de faciliter la structuration de la documentation d'un logiciel, ce qui permet d'éviter tout risque

d’ambiguïté ou de confusion lors de la spécification des besoins. Finalement, elle soutient l’automatisation du processus de vérification de cohérence réduisant ainsi les coûts de maintenance.

4.2. Ontologies dans le Web sémantique

Avec l’apparition du fameux web sémantique, les ontologies ont joué un rôle technologique clé pour représenter et stocker les informations à travers des modèles qui préservent les sémantiques adressées par les domaines des applications porteuses. Ceci facilite d’une part, l’indexation et la recherche d’information sur le web sémantique et améliore considérablement la qualité des résultats obtenus, à travers deux points majeurs:

- Une ontologie permet à l’utilisateur d’utiliser un autre terme que celui présent dans les documents.
- Une ontologie permet au système de recherche d’information d’aider l’utilisateur à reformuler ses requêtes sur la base d’une proximité sémantique.

D’autre part, elle permet de faire des traitements automatiques à des modules logiciels au sein du Web sémantique, que ce soit pour faire interagir et interopérer des machines entre elles ou faire interagir des machines avec des humains.

D’un autre côté, grâce au Web sémantique, l’ontologie a trouvé un jeu de formalismes standards à l’échelle mondiale, et s’intègre dans de plus en plus d’applications Web. Cela se fait au profit des logiciels qui, à travers les ontologies et les descriptions qu’elles permettent, peuvent proposer de nouvelles fonctionnalités pour en améliorer les effets.

5. Classification des ontologies

En survolant la littérature des ontologies, plusieurs critères peuvent être utilisés pour classifier les ontologies. Dans cette section nous allons présenter les critères les plus cités, en l’occurrence: l’objet de la conceptualisation, le niveau de granularité, le niveau de formalisme de représentation, et le poids de l’ontologie. Nous illustrons dans la figure 1.1, une classification des ontologies selon les quatre critères ainsi cités:

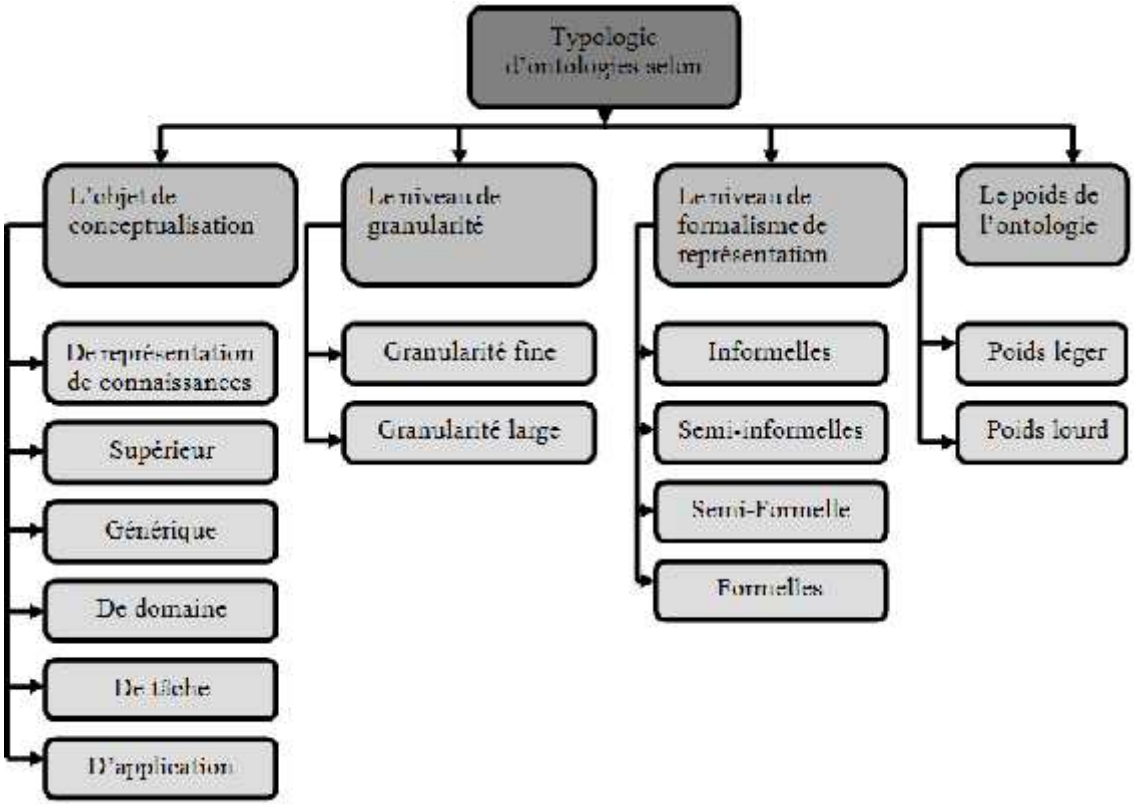


Figure 1-1. Critères de classification d'ontologies

5.1. Classification selon l'objet de conceptualisation:

Selon l'objet de conceptualisation, les ontologies sont classifiées en six catégories :

- Ontologie de représentation des connaissances**:** Elle se focalise sur la représentation des classes impliquées dans la formalisation des connaissances. (Gómez-Pérez, 1999a).
- Ontologie supérieure: Son objectif est l'étude des catégories des choses qui existent dans le monde tels que: les entités, les relations, les propriétés, le temps, l'espace. (Guarino, 1995; Sowa, 1995).
- Ontologie Générique : Elle représente des connaissances génériques mais assez générales pour être réutilisées à travers différents domaines. (Gómez-Pérez, 1999a)
- Ontologie du Domaine : Elle caractérise la connaissance du domaine où la tâche est réalisée. (Définie par Mizoguchi, en 2000).
- Ontologie de Tâches : Ce type d'ontologies est utilisé pour conceptualiser des tâches spécifiques dans les systèmes, telles que les tâches de diagnostique, de planification, de conception, de configuration, etc. (Mizoguchi, 2003).

- Ontologie d'Application : Il s'agit de l'ontologie la plus spécifique où les classes correspondent souvent aux rôles joués par les entités du domaine tout en exécutant une certaine activité, (Maedche & Staab, 2002).

5.2. Classification selon le niveau de granularité

Selon le niveau de granularité (niveau de détail), les ontologies sont classifiées en :

- Ontologies de fine granularité
- Ontologies de large granularité

5.3. Classification selon le niveau de formalisme de représentation

Selon le niveau de formalité du langage de représentation, les ontologies sont classifiées par (Uschold & Gruninger, 1996) en :

- Ontologies informelles (représentées en langage naturel).
- Ontologies semi-informelles (représentées en langage naturel structuré et limité).
- Ontologies semi-formelles (représentées en langage artificiel défini formellement).
- Ontologies formelles (représentées en langage artificiel avec sémantique formel, DL).

5.4. Classification selon le poids de l'ontologie

Selon leur poids, les ontologies sont classifiées par (Mizoguchi, 2003), en :

- Ontologies de poids léger (classes et relations)
- Ontologies de poids lourd (classes, propriétés et relations)

6. Ontologies linguistiques Vs Ontologies formelles:

Les ontologies linguistiques sont des ressources lexicales vastes qui couvrent la plupart des mots d'un langage. Elles offrent une structure ontologique qui se concentre principalement sur les relations entre les classes. Les ontologies linguistiques peuvent alors être vues comme étant une forme particulière d'une base de données lexicale mais aussi comme étant une forme particulière d'ontologies. De plus, les ontologies formelles sont simplement des ontologies qui se basent sur l'utilisation d'un langage artificiel qui contient une sémantique formelle telle que la logique de description. Ce qui diffère principalement une ontologie linguistique (tel que wordNet) d'une ontologie formelle (tel que SENSUS) est son degré de formalisation. En effet, les

ontologies linguistiques ne reflètent pas les aspects formels reflétés par les ontologies formelles. Par exemple, l'ontologie linguistique wordNet ne présente aucune distinction entre les types et les rôles et sa structure hiérarchique ne reflètent aucune information de disjonction mutuelle entre les classes. Un autre critère, qui fait la distinction entre les ontologies linguistiques et les ontologies formelles, est leurs tailles. En effet, les ontologies linguistiques sont beaucoup plus larges que les ontologies formelles. En outre, les avantages des ontologies linguistiques sont reflétés par leurs caractéristiques principales. Au niveau linguistique, les ontologies linguistiques sont fortement dépendantes des langages naturels. Elles sont basées sur les mots les plus fréquemment utilisés dans un (ou plusieurs) langage spécifique (ontologie mono ou multilingues). An niveau sémantique, et à l'instar des ontologies formelles, les ontologies linguistiques supportent le partage des classes décrites par différents mots dans différents langages. Vu que les ontologies linguistiques couvrent des informations ontologiques mais aussi linguistiques, elles offrent des solutions partielles aux limitations des ontologies formelles (qui présentent des couvertures limitées des domaines d'application et donc qui doivent être mises à jour périodiquement) lorsqu'on veut les utiliser dans un système de recherche d'informations (IR). En 2001, les ontologies linguistiques ont été utilisées dans le contexte des technologies des agents distribués, où le problème de négociation des sens est primordial (Bouquet & Serafini, 2001).

7. L'ingénierie ontologique:

Dans cette section, nous allons répondre aux questions de spécification qu'on doit savoir avant de construire une ontologie: Quelles sont les principes qu'on doit suivre pour construire une ontologie ? Quelle méthodologie doit-on utiliser pour construire sa propre ontologie ? Quelles sont les étapes qu'on doit suivre lors d'un processus de construction d'ontologie et quelles sont les phases à aborder par son cycle de vie ?

7.1. Les principes de construction d'ontologies

Pour mieux guider le processus de construction d'ontologies, plusieurs critères et principes de base ont été proposés dans la littérature, notamment par (Bachimont, 2001;

Gruber, 1993). Dans (Gruber, 1993), Cinq critères génériques guidant l'ontologisation ont été proposés :

- **La clarté et l'objectivité:** Les termes doivent être définis de façon claire et objective et disposant d'une documentation en langage naturel.

õ **La cohérence:** Les axiomes doivent être consistants afin de permettre à l'ontologie de réaliser des inférences cohérentes aux définitions.

õ **Extensibilité monotone maximale:** C'est-à-dire, la possibilité d'ajouter de nouveaux termes et d'étendre l'ontologie sans avoir à réviser ou à modifier les anciennes.

õ **Minimalité des postulats d'encodage assurant une bonne portabilité.**

õ **Engagement ontologique minimal:** Il s'agit de donner aux utilisateurs d'ontologie une marge de liberté plus grande pour spécifier et instancier l'ontologie selon leurs besoins indépendamment des suppositions concernant le monde modélisé.

Dans (Bachimont, 2001), quatre critères en plus ont étaient proposés :

õ **Le principe de communauté avec le père, ou principe de similarité:** C'est-à-dire que la classe hérite l'intention de sa classe père.

õ **Le principe de différence avec le père, ou le principe de différence:** C'est-à-dire que l'intention d'une classe doit être différente de celle de son père.

õ **Le principe de communauté avec la fratrie ou principe de sémantique unique:** C'est-à-dire qu'une propriété qui est commune entre les classes frères ayant le même père mais elle est exprimée différemment pour chaque frère.

õ **Le principe de différence avec la fratrie ou principe d'opposition:** C'est-à-dire que les frères doivent tous être incompatibles, sinon il n'aurait pas lieu de les définir.

7.2. Méthodologies de construction d'ontologies :

Selon le type de construction qu'il va adopter (construire une ontologie à partir de zéro, ou construire une ontologie par intégration ou utilisation des ontologies préalablement existantes tels que la fusion ou l'alignement), le concepteur peut suivre une des méthodologies de construction d'ontologies, les plus connues:

7.2.1. La méthodologie Entreprise (Uschold & King, 1995)

Elle s'est inspirée des constructions modélisant une entreprise. Elle comprend l'identification de l'objectif de l'ontologie, sa construction, son évaluation, et

finalement sa documentation. Lors du processus de construction, les auteurs proposent de capturer les connaissances, les coder, et finalement intégrer si nécessaire des ontologies préexistantes dans l'ontologie courante. Les trois stratégies suivantes sont proposées par les auteurs pour identifier les classes :

- Une stratégie descendante (top down strategy) où les classes les plus générales sont identifiées puis sont spécialisées,
- Une stratégie ascendante (down top strategy) où les classes les plus spécifiques sont identifiées puis sont généralisées,
- Une stratégie mixte (midle out strategy), où les classes les plus importantes sont identifiées puis sont généralisées et/ou spécialisées. Cette méthodologie est indépendante du système qui l'exploite et son processus de construction d'ontologie est indépendant de l'objectif de l'ontologie.

7.2.2. La méthodologie TOVE (Gruninger & Fox, 1995)

Elle s'est inspirée du développement des SBC utilisant la logique du premier ordre. Elle comprend:

- L'identification des principaux scénarios (les applications de l'ontologie),
- L'identification des questions de compétence (que le système est censé pouvoir répondre) et alors identifier les classes et les axiomes de l'ontologie en utilisant une stratégie mixte (midle out).

Dans cette méthodologie qui est semi dépendante du système qui l'exploite, une description informelle des spécifications de l'ontologie sont formalisés (logique de premier ordre), donc elle peut être utilisée comme un guide de transformation de scénarios informels en modèles formels.

7.2.3. La méthodologie Methontology (Gómez-Pérez, Fernández, & De Vicente, 1996)

Cette méthodologie est utilisée pour construire une ontologie que ce soit à partir de zéro, ou en réutilisant d'autres ontologies préexistantes et elle comprend: L'identification du processus de développement d'ontologie, c'est-à-dire qu'elle couvre tout le cycle de vie de l'ontologie qui est basé sur un prototype évolutif et sur des techniques particulières pour la réalisation des activités liées aux ontologies qui sont: l'ordonnancement de contrôle, (scheduling control), assurance de qualité,

spécification, acquisition de connaissances, conceptualisation, en plus des activités de gestion de projets, et de support. (c) Cette méthodologie est indépendante de l'application qui l'exploite. Elle utilise une stratégie mixte pour l'identification des classes de l'ontologie.

7.3. Le processus de construction d'ontologies

Les ontologies sont considérées comme étant des objets techniques évolutifs ayant un cycle de vie. Leur processus de développement passe en général par les mêmes étapes de développement d'un logiciel qui à partir des données brutes, exprimées généralement en langages naturels (informels), développe le modèle conceptuel dans une étape de conceptualisation. Puis, il génère l'ontologie formelle dans une étape d'ontologisation qui va l'exprimer dans un langage de représentation formel et opérationnel lors de l'étape de l'opérationnalisation.

Il est à noter que le processus de construction ontologique n'est pas purement linéaire car la réalisation d'une étape peut mettre en cause les résultats des étapes précédentes, d'où la nécessité de retours en arrière pour supprimer, ajouter ou modifier les modules antérieurs. La Figure 1.2 représente un schéma général du processus de construction d'ontologie.

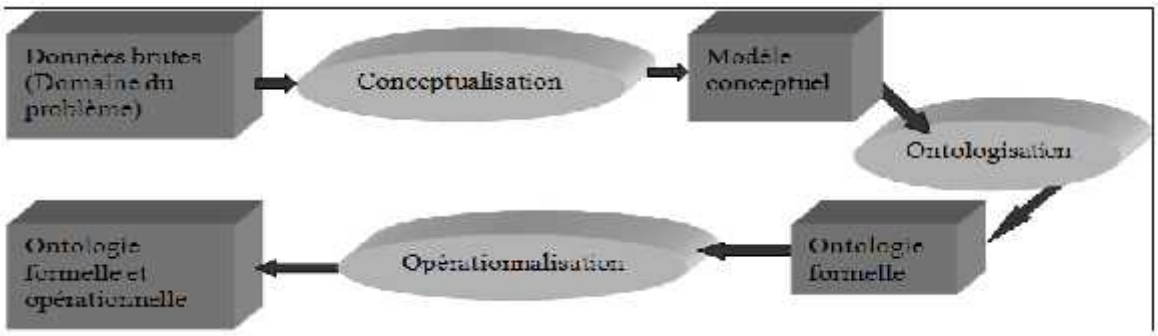


Figure 1-2. Processus de construction d'ontologie

7.3.1. La conceptualisation:

À partir d'un corpus de données brutes, la conceptualisation consiste à identifier les connaissances du domaine d'application, tout en dégageant les classes et les relations entre elles. Le processus de la conceptualisation doit faire la distinction entre les connaissances spécifiques au domaine et celle ne servant qu'à l'expression des connaissances de domaine. En plus, si le développement ontologique prévoit l'intégration d'autres ontologies, il ne faut pas prendre en compte les connaissances

déjà présentées dans ces ontologies dans la conceptualisation de l'ontologie en cours. Ensuite, la nature conceptuelle des termes ontologique désignant les entités du domaine de connaissances (tels que classes, relations, axiomes, propriétés, etc.) doit être précisée. L'acquisition des connaissances du domaine demande l'intervention d'un expert (ou d'un groupe d'experts) du domaine assisté par l'ingénieur de connaissances qui applique son expertise des techniques de représentation de connaissances pour les rendre machinables, favorisant ainsi la structuration de connaissances pour aboutir à un modèle conceptuel composé de classes, relations, axiomes, propriétés, slots, facets, etc.

Une des tâches les plus délicates dans le processus de conceptualisation est d'identifier les connaissances qui sont implicitement utilisées dans le domaine, mais qui n'ont pas été exprimé ni dans le corpus analysé, ni par les experts du domaine car, par exemple, elles semblent complètement évidentes, d'où vient l'importance des rôles des utilisateurs finaux dans le processus de la construction ontologique, et spécialement en phase de la conceptualisation pour la mise en évidence de ces connaissances implicites à travers l'utilisation de l'ontologie, lors d'une phase de test opérationnel et/ou de test de questions de compétence. Ce qui prouve l'assertion précédemment citée, que le processus de construction n'est pas purement linéaire où un retour en arrière aura lieu lors de la découverte de connaissances non spécifiées par l'utilisateur final de l'ontologie.

7.3.2. L'ontologisation:

Cette étape consiste à structurer et à formaliser autant que possible (partiellement) le modèle conceptuel résultant de la phase précédente pour construire une ontologie spécifiant la terminologie et la sémantique du domaine à travers un modèle doté d'une sémantique formelle mais pas opérationnelle. Cette formalisation partielle, menée par l'ingénieur de connaissance assisté par l'expert de domaine, facilite sa représentation ultérieure dans un langage complètement formel et opérationnel. En d'autres termes, l'ontologisation est vue comme étant une traduction dans un certain formalisme de représentation des connaissances exprimées à priori en langage naturel (Furst, 2004). A ce stade, les connaissances ne sont pas toutes totalement formalisées vue l'impossibilité de lever certaines ambiguïtés et/ou la difficulté ou l'impossibilité de

formaliser certaines expressions en langage naturel donnant ainsi le caractère semi formel à cette ontologie, ce qui empêchera (par exemple) son intégration dans un système à base de connaissances.

7.3.3. L'opérationnalisation:

Elle consiste à transcrire l'ontologie (semi) formelle, résultante de l'étape précédente, dans un langage formel et opérationnel de représentation d'ontologies. A ce stade, les termes ontologiques (classes et relations) sont enrichis par les opérations qu'on peut appliquer pour inférer ou déduire de nouvelles connaissances permettant des interprétations possibles menées par la représentation formelle, et aussi de spécifier la façon par laquelle ces connaissances sont exploitées pour inférer ou déduire de nouvelles connaissances. En d'autres termes, l'opérationnalisation permet de spécifier comment ces connaissances opèrent sur d'autres pour faire un raisonnement, ce qui rend l'ontologie intégrable et utilisable par un SBC. Il est à noter que le processus d'opérationnalisation est mené par l'ingénieur de connaissances.

Nous pouvons donc résumer que la nature structurelle des représentations de connaissances résultantes des trois phases du processus de construction ontologique sont comme suit:

La conceptualisation (informelle ou semi formelle): En langage naturel ou semi structuré, ne disposant pas de sémantique claire ou tout au moins d'une sémantique fixée à priori.

L'ontologisation (formelle): Spécifie la syntaxe et la sémantique.

L'opérationnalisation (formelle et opérationnelle): Dotée de services d'inférences permettant de mettre en œuvre des raisonnements.

7.4. Cycle de vie d'une ontologie

Comme tout composant logiciel, une ontologie est destinée à être réutilisée et parfois partagée entre plusieurs applications (et/ou individus d'une communauté) satisfaisant des besoins et des fonctions opérationnels différents. Alors pour construire une ontologie, on peut adopter pour les mêmes techniques et principes que ceux appliqués en GL lors du développement des Systèmes d'Information.

Selon (Saïd, 2006), les activités liées aux ontologies sont d'une part des activités de développement (Spécification, Formalisation et Implémentation), et d'autre part des activités de gestion de projet (Organisation, Planification, Estimation, Contrôle et

Assurance de qualité), s'y ajoute un certain nombre d'activités transversales de support (Evaluation, Documentation, gestion de la configuration). Plusieurs spécifications différentes de cycles de vie d'ontologies inspirées du GL ont été proposées dans la littérature, mais qui se ressemblent toutes dans les phases principales de cycle de vie.

Un cycle de vie qui constitue les phases les plus communes entre les différentes propositions est illustré dans la figure 1.4:

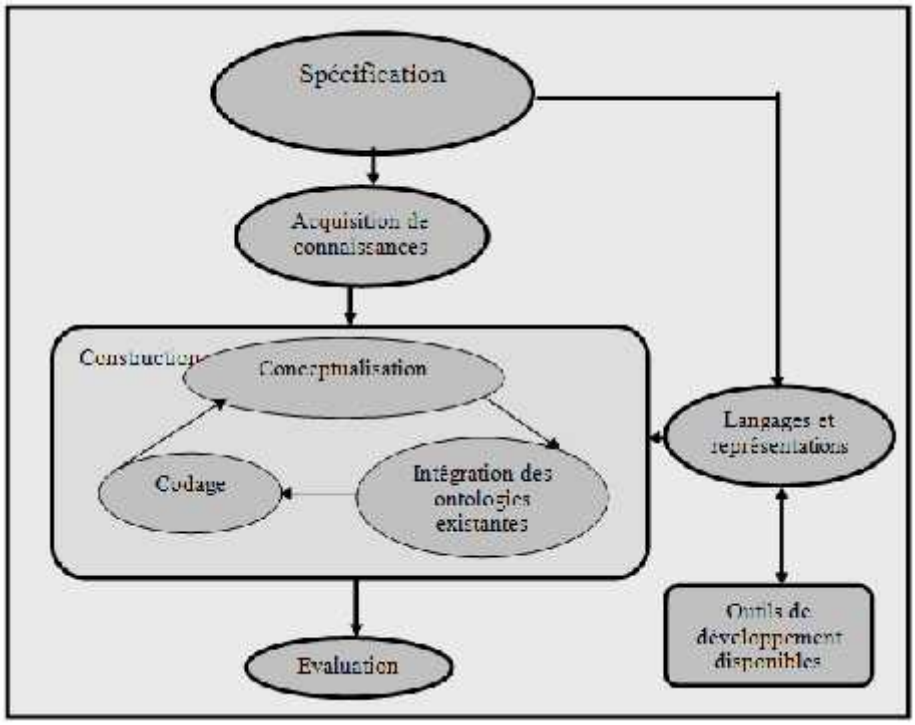


Figure 1-3. Cycle de vie de développement d'ontologie.

Nous pouvons décrire les étapes illustrés en Figure 1.4 comme suit:

7.4.1. Spécification: C'est une étape primordiale pour la conception, l'évaluation et la réutilisation de l'ontologie. Elle consiste à identifier les propositions, les niveaux de granularité et de généralisation des classes .

7.4.2. Acquisition de connaissances: Elle consiste en l'absorption et la collection de connaissances de domaine à travers l'analyse des scénarios et documents tel que les BDDs, les textes standardisés, les papiers scientifiques, en plus d'autres ontologies, ou en organisant des interviews avec les experts de domaine qui doivent être accompagnés par un ingénieur de connaissances.

7.4.3. La conceptualisation: Elle consiste en l'identification des classes clés reflétant le domaine de connaissances, leurs propriétés, ainsi que les relations qui les relient puis en la mise en évidence des descriptions de ces classes, relations et attributs en langage naturel et enfin en la modélisation des connaissances de domaines à travers un modèle conceptuel explicite. Cette étape doit être menée par l'expert de domaine assisté par l'ingénieur de connaissances.

7.4.4. Intégration: Elle consiste en la combinaison des données (et/ou connaissances) valables à partir de BDDs, et ontologies précédemment construites pour obtenir une ontologie consistante et plus complète.

7.4.5. Codage: Elle consiste en la représentation du modèle conceptuel dans un langage formel de représentation ontologique tels que la logique de description, les frames, les réseaux sémantiques, etc.

7.4.6. Evaluation: Elle consiste en l'appréciation de compétence et capacité de l'ontologie à satisfaire les besoins de son application. Il s'agit donc d'évaluer l'ontologie du point de vue de la complétude, la consistance, et la minimalité de redondances.

7.4.7. Documentation: Des descriptions et des définitions complètes, formelles ou informelles des assomptions et des exemples sont inévitables pour faciliter et aider à l'utilisation ou la réutilisation des ontologies, car une ontologie qui ne peut être comprise ne peut être utilisée.

8. Les erreurs de construction d'ontologies:

L'évaluation du contenu ontologique est une phase critique du cycle de vie de la construction d'ontologies car si l'ontologie souffre de certaines erreurs, les applications qui les utilisent peuvent avoir des problèmes critiques et catastrophiques et donc l'ontologie peut ne pas atteindre son objectif (Fahad, Qadir, & Noshairwan, 2007). Les ontologies peuvent alors être évaluées en considérant des principes de conception (Fernandez-Lopez & Corcho, 2010; Gómez-Pérez, 1995, 1999b), les contraintes logiques de validation des axiomes, relations, instances, etc., l'applicabilité de l'ontologie dans un domaine particulier (Porzel & Malaka, 2004), les prédictions de leurs résultats, sa comparaison avec d'autres ressources de données (Maedche &

Staab, 2002). Parmi tous ces travaux et d'autres la classification la plus succincte, pratique et générique est celle proposée par (Asunción Gómez-Pérez, 1999b; Fernandez-Lopez & Corcho, 2010). En considérant des principes de conception, Gomez-Perez a identifié trois classes principales des erreurs d'hiérarchie ontologique, en l'occurrence, les erreurs d'inconsistance, d'incomplétude et de redondance :

8.1. Les erreurs d'inconsistance

Trois différentes formes d'erreurs peuvent causer des problèmes d'inconsistance et d'ambigüité :

8.1.1. Erreurs de circuit

Ce type d'anomalie apparaît quand une classe est définie comme étant une sous classe ou super classe d'elle-même. Ceci peut apparaître à n'importe quel niveau de l'hiérarchie. Il peut apparaître avec une distance de 0 ou de n, selon le nombre de relations constituant le chemin à traverser pour atteindre la même classe de départ.

8.1.2. Erreurs de partitionnement

Ils existent différentes manières de classification, en dépendant du type de décomposition des superclasses en sous classes. Quand tous les attributs des sous classes sont décrits d'une manière indépendante l'un de l'autre et il n y a pas d'intersection entre les sous classes alors on dit que la décomposition est disjointe. Quand les ontologies suivent des contraintes de complétude entre les sous-classes et les super classes alors on dit que la décomposition est complète ou exhaustive. De ces points, on dit qu'il y a deux différents types d'erreurs de décomposition :

8.1.2.1. Des classes et instances communes dans une décomposition (partitionnement) disjointe

Ce type d'anomalies apparaît quand les ontologistes créent des instances qui appartiennent aux différentes sous classes disjointes. Ou quand ils créent une classe commune comme étant une sous classe commune de deux supers classes disjointes.

8.1.2.2. Des instances externes dans une décomposition exhaustive

Ce type d'anomalies apparaît quand les ontologistes effectuent une décomposition (partitionnement) exhaustive d'une classe en plusieurs sous classes mais ce ne sont pas toutes les instances de la classe de base appartiennent à l'une des sous classes. i.e. une instance ou plus de la classe de base n'appartient pas à aucune sous classe.

8.1.3. Les erreurs d'inconsistance sémantique

Ce type d'anomalies apparaît quand les ontologistes font une hiérarchie incorrecte de classes en classifiant un classe comme étant une sous classe d'une autre classe qui ne l'appartient pas réellement. Par exemple, classifier une voiture comme étant une sous classe d'un avion. Le même problème peut apparaître lors de la classification des instances. Dans (Fahad & Qadir, 2008), les auteurs ont distingué entre trois différents types d'erreurs d'inconsistance sémantique, en l'occurrence :

8.1.3.1. L'erreur de spécification d'un domaine plus large par une sous classe :

Ce type d'anomalies apparaît quand les classes qui représentent un domaine plus large sont spécifiées comme étant des sous classes d'une classe qui représente un domaine plus restreint. Par exemple, l'ontologiste classifie les classes *université*, *écoleSupérieur* et *institut* comme étant des sous classes de la classe *département*. Une sous classe doit toujours spécialiser (être subsumée par) les propriétés de la classe de son super classe en spécifiant un domaine plus restreint et en rendant le domaine de la super classe plus spécifique.

8.1.3.2. L'erreur de spécification d'une rupture du domaine par une sous classe

Les sous classes doivent posséder toutes les propriétés de la classe parent et ne doivent jamais violer aucune propriété de leurs parents dans leurs propres domaines. La rupture du domaine de la super classe apparaît quand la classe spécifiée par la sous classe ajoute plus de propriétés qui ne se présentent pas dans la super classe, mais que les propriétés additionnelles violent quelques propriétés de leurs super classes. Par exemple, dans une hiérarchie de classe « pizza », l'ontologiste classifie la classe « vegetarianPizza » comme étant une sous classe de la classe « pizza ». Ensuite, les classes « chineesePizza » et « italianPizza » sont classifiées comme étant des sous classes de la classe « vegetarianPizza ». L'inconsistance sémantique apparaît vu que la définition de « chineesePizza » implique l'inexistence d'aucune pizza préparée par des produits végétaux ou viande.

8.1.3.3. L'erreur de spécification des classes disjointes par des sous classes

Ce type d'anomalies apparaît quand l'ontologiste spécifie des classes disjointes d'un domaine comme étant des sous classes d'une classe qui occupe un domaine différent. Par exemple, il classifie les classes « boisson » et « burger » comme étant des sous

classes de la classe « produitMangeable », mais aucune propriété de la classe « boisson » ne correspond à la super classe de la classe « produitMangeable ». i.e. ils appartiennent à des domaines disjoints.

8.2. Les erreurs d'incomplétude

Parfois les ontologistes effectuent une classification des classes mais oublient quelques informations importantes. Cette incomplétude crée parfois des ambigüités et élimine des mécanismes de raisonnement. Deux différents types d'erreurs d'incomplétude peuvent apparaître :

8.2.1. Classification incomplète des classes

Ce type d'erreurs apparaît quand l'ontologiste oublie quelques classes du domaine lors de la classification d'une classe particulière. Par exemple, lors de la classification d'une classe « location » en sous classes « locationCulturelle », « locationMontagne », il oublie autre type de location telles que « locationPlage », « locationHistorique », etc.

8.2.2. Erreur de partition incomplète

Quand l'ontologiste oublie des axiomes ou des informations importantes concernant la classification d'une classe, ceci réduit le potentiel des raisonnements et des mécanismes d'inférence. Gomez-Perez a identifié deux types d'erreurs conduisant à une partition incomplète :

8.2.2.1. L'oubli des connaissances disjointes

Cette erreur apparaît quand l'ontologiste classifie une classe en plusieurs sous classes et partitions mais qu'il oublie des axiomes de connaissances disjointes entre elles. Par exemple, l'ontologiste modélise « locationPlage », « locationMontagne » et « locationHistorique » comme étant des sous classes de la classe « location » mais il oublie de modéliser l'axiome de connaissances disjointes entre les sous classes.

8.2.2.2. L'oubli des connaissances exhaustives

Ce type d'erreurs apparaît quand l'ontologiste ne suit pas les contraintes de complétude lors de la décomposition d'une classe en sous classes et partitions. Par exemple, l'ontologiste modélise les classes « locationPlage », « locationMontagne » et « locationHistorique » comme étant disjointes mais il ne spécifie pas si cette classification forme ou non une décomposition exhaustive. Les étiquettes « functional » et « inverse functional » associées aux propriétés indiquent combien de

fois une classe source peut être associée à un classe cible à travers une propriété. Parfois, l'ontologiste ne donne aucune signification à ces étiquettes de propriétés. Alors la machine ne pourra pas raisonner sur ces propriétés ce qui conduit à des complications sérieuses (Qadir, Fahad, & Shah, 2007).

Dans (Fahad & Qadir, 2008), les auteurs ont proposé une extension des erreurs de complétude par trois autres formes de problèmes d'incomplétude :

8.2.3. Oubli d'une propriété fonctionnelle pour une propriété mono valeur :

Selon le méta modèle de définition d'ontologies (Deere, 2005), quand il y a une seule valeur pour un sujet donné, alors cette propriété a besoin d'être déclarée comme étant « fonctionnelle ». L'étiquette « fonctionnelle » peut être associée avec la propriété d'objets mais aussi avec la propriété de type de données. Par exemple, la propriété d'objet « aPourGroupage » entre « personne » et « groupageDeSang » est un exemple d'une propriété d'objet fonctionnelle. Chaque sujet « personne » n'appartient uniquement qu'à un seul type de « groupageDeSang », alors, la propriété « aPourGroupage » doit être étiquetée comme étant « fonctionnelle », et donc le sujet « personne » ne doit être associé qu'à un seul « groupageDeSang ».

De même, les propriétés de type de données fonctionnelles accordent un seul rang R pour chaque instance de domaine D. L'ignorance de l'étiquette « fonctionnelle » permet à la propriété d'avoir plus d'une seule valeur, ce qui conduit aux inconsistances. La raison principale de cette inconsistance est que l'ontologiste ignore que les ontologies owl supportent, par défaut, la multiplicité de valeurs pour les propriétés d'objets et les propriétés de type de données.

8.2.4. Oubli d'une propriété inverse fonctionnelle pour une propriété mono valeur

Selon le méta modèle de définition d'ontologies (Deere, 2005), une propriété inverse fonctionnelle d'un objet détermine le sujet d'une manière unique. i.e. Elle joue le rôle d'une clé dans une BDD. Cela veut dire que si on prend p est une propriété inverse fonctionnelle, ceci restreint que pour une instance unique on peut trouver une valeur unique « x ». i.e. on ne peut pas trouver deux instances différentes y et z tel que les deux couples (y, x) et (z, x) sont des instances valides de p. Le « numSécuritéSocial » est un exemple d'une propriété de type de données inverse fonctionnelle appartenant à

« personne » vu qu'elle identifie « personne » d'une manière unique. Ignorer l'étiquette « inverse fonctionnelle » avec la propriété « numSécuritéSocial » crée une inconsistance dans l'ontologie grâce à la spécification incomplète de la classe. Une telle ignorance ne permet pas à une machine d'inférer et de raisonner sur une classe d'une manière unique.

8.2.5. L'erreur d'oubli d'une connaissance suffisante

Les classes dans une hiérarchie ontologique doivent avoir certains critères permettant au moteur d'inférence d'en distinguer l'un de l'autre d'une façon appropriée. En logique de description, à chaque classe est associée une description nécessaire et une description suffisante (Baader et al., 2003). Les règles de description nécessaire définissent les critères de base par lesquels une nouvelle classe est formée par la sous classe d'une relation. La description suffisante définit une classe en terme d'autres classes par l'intersection, l'union, le complément ou les axiomes de restriction en owl (Noshairwan, Qadir, & Fahad, 2007). Parfois, et lors de la conception d'ontologies, la description définit les classes mais ne fournit pas les descriptions suffisantes, raison de quoi la machine ne peut pas raisonner sur eux (d'une façon appropriée) et donc ne peut pas les utiliser pour atteindre les objectifs du web sémantique. Une des façons de détection automatique des incomplétudes dans une ontologie est d'évaluer l'ontologie en utilisant des données de test (valides ou non valide), (Brewster et al., 2004).

8.3. Les erreurs de redondances

Les redondances appariaient quand une information particulière est inférée plus qu'une fois. Différents types d'erreurs de redondances sont mis au point :

8.3.1. Redondances des relations SubClassOf, SubPropertyOf et InstanceOf

Les erreurs de redondance des relations SubClassOf appariaient quand l'ontologiste spécifie des classes ayant plus d'une relation SubClassOf directement ou indirectement. Directement signifie que la relation SubClassOf est entre une classe et sa super classe directe. Indirectement signifie que la relation SubClassOf et entre une classe et sa super-classe indirecte à n'importe quel niveau. Par exemple l'ontologiste spécifie « LocationPlage » comme étant une sous classe de « location » et aussi sous classe de « place », et par la suite, il spécifie « location » comme étant une sous classe de « place ». Ici, il y a une relation SubClassOf indirecte entre « locationPlage » et

« place » créant ainsi une redondance. De même, les redondances de relations SubPropertyOf peuvent apparaître lors de la construction des hiérarchies de propriétés. Les redondances des relations InstanceOf appariaient quand l'ontologiste spécifie, par exemple, l'instance « swat » comme étant une instance des deux classes « location » et « place », et il est aussi défini que « location » est sous classe de « place ». La relation explicite InstanceOf entre « swat » et « place » crée une redondance vu que « swat » est une instance indirecte de « place » et « place » est une super classe de « location ».

8.3.2. Définition formelle identique de classes, propriétés et instances

Ce type d'erreur apparaît quand l'ontologiste définie différents noms (ou même des noms similaires) de deux classes, propriétés ou instances respectivement mais fournit la même définition formelle.

Une autre forme d'erreur de redondance a été suggérée dans (Fahad & Qadir, 2008):

8.3.3. L'erreur de redondance de relation de disjonction

Ce type d'erreur apparaît quand une classe est explicitement définie comme étant disjointe d'une autre classe plus d'une seule fois (Noshairwan et al., 2007). Selon les règles de la logique de description (Baader et al., 2003), si une classe est disjointe d'une autre classe alors elle est disjointe de toutes ses sous classes. Le cas unique d'apparence de ce type d'erreur est quand une classe est explicitement spécifiée comme étant disjointe d'une classe parent mais aussi de ses sous classes. Ce type d'erreur peut apparaître grâce à une relation de disjonction directe mais aussi indirecte qui se répète.

9. L'évaluation d'une ontologie

Comme tout processus d'évaluation, l'évaluation d'ontologie consiste en deux sous processus complémentaires, à savoir la vérification qui consiste à vérifier qu'on a bien construit l'ontologie selon le modèle formel spécifié, et la validation qui consiste à s'assurer qu'on a construit la bonne ontologie relativement au domaine d'application.

9.1. La vérification:

Elle consiste à vérifier si l'ontologie construite est correcte, c'est à dire qu'elle correspond bien au modèle conceptuel représentant le domaine d'application. Selon

(Furst, 2004), la vérification d'une ontologie repose sur le test de trois grands types de propriétés, en l'occurrence: la conformité, la cohérence, et la minimalité :

- **La conformité d'une ontologie à un modèle conceptuel:** Il s'agit de vérifier que la forme de l'ontologie est conforme à la syntaxe imposée par le modèle conceptuel (indépendamment du domaine d'application). Vu que le test de conformité ne vérifie que la syntaxe, on peut trouver des axiomes qui sont syntaxiquement corrects, mais qui peuvent être logiquement contradictoires.

õ **La cohérence:** Il s'git de s'assurer que les connaissances représentées dans l'ontologie ne présentent pas de contradictions logiques (et/ou sémantiques). Il est à noter que le test de cohérence est aussi indépendant du domaine d'application.

õ **La minimalité:** Il s'agit de vérifier que l'ontologie a la taille la plus petite que possible dans le sens où elle ne contient pas des connaissances superflues.

En résumé, le processus de la vérification vise à contrôler qu'il n'y a pas de cycles dans les hiérarchies constituant l'ontologie, qu'il n'y a pas de redondances de classes ou de relations et que ces hiérarchies sont bien connexes (c'est à dire qu'elles ne présentent pas des parties isolées des autres, et donc sans aucun sens, vu que la sémantique est transmise à travers les liens).

9.2. La validation:

Elle consiste à s'assurer qu'on a construit la bonne ontologie qui doit être conforme au domaine de connaissances dans le sens où elle reflète toute la sémantique qu'il renferme, et complète c'est à dire qu'elle présente toutes les connaissances du domaine qu'elle modélise.

En général, le processus de la validation consiste à poser des questions de compétence dans un système à base de connaissances (SBC) opérationnel. L'impossibilité de poser de telles questions ou d'en fournir une bonne réponse ou la possibilité de fournir une réponse mais qui n'est pas correcte ou cohérente, met en cause la modélisation des connaissances et nécessite un retour en arrière pour une éventuelle correction ou maintenance de l'ontologie.

10. Langages et éditeurs d'ontologies

10.1. Les langages de représentation d'ontologies

Le codage et l'expression d'une ontologie en utilisant un langage de représentation formel permettent de l'exploiter par une application. A l'aube des années 90 plusieurs langages de représentation d'ontologies basés sur l'IA et s'inspirant essentiellement de la logique de premier ordre, la logique de description ou les frames de Minsky ont été spécifiés. Au milieu des années 90, et avec l'émergence d'Internet d'autres langages basés web ont été créés permettant d'exploiter la richesse du web.

Dans ce qui suit, nous allons présenter une synthèse des langages de représentation des ontologies les plus cités dans la littérature :

10.1.1. Les langages de représentation traditionnels:

- **KIF** (Knowledge Interchange Format): Introduit en 1992, il se base sur la logique de premier ordre pour représenter les classes, les relations, les fonctions, les axiomes, les instances ainsi que les procédures. Il est considéré comme le langage le plus expressif par rapport aux autres langages de représentation d'ontologies.

http://logic.standford.edu/kif/kif.html

- **Le langage ONTOLINGUA***:* Créé en 1992, à l'université de Standford en ajoutant une syntaxe au langage KIF permettant de capturer intuitivement l'assemblage des axiomes en combinant des frames de la logique du premier ordre, supportant ainsi la conception et la spécification d'ontologies à travers un logique sémantique (basé KIF) en modélisant les relations, les classes, les fonctions, les individus et les axiomes.

http://ontolingua.standford.edu/

- **LOOM:** Créé en 1992 à l'université de Southcalifornia, il est basé sur la logique de description et les règles de production. Il peut fournir une classification automatique des classes à travers un moteur déductif (classifieur) raisonnant en chaînage avant et fournissant une unification sémantique.

http://www.isi.edu/isd/loom/loom-home.html

- **OCML** (Operational Conceptual Modelling Language): Créé en 1993 par KMI de l'Open University, en ajoutant des définitions opérationnelles pour chaque fonction du langage ONTOLINGUA, et en prenant en charge les règles de production et de déduction facilitant ainsi la construction de mécanismes de raisonnement, ce qui fait

d'OCML un langage de représentation et de développement d'ontologies exécutables adaptées aux méthodes de résolution de problèmes.

- **FLOGIC** (Frame-Logic): Créé en 1995 à l'université de Karlsruhe, il combine la logique de premier ordre et les frames de Minsky pour représenter les classes, les taxonomies de classes, les relations binaires, les fonctions, les instances, les axiomes et les règles de déduction, ces dernières sont utilisées par le moteur d'inférence (OntoBroker) pour dériver de nouvelles connaissances.
- **OKBC** (Open Knowledge Base Activity): Créé en 1997, il s'agit d'un protocole plutôt qu'un langage de représentation. Il permet l'accès aux ontologies et base de connaissances à travers un API (Application Program Interface) intégré. OKBC offre un modèle uniforme de système de représentation de connaissances basé sur une conceptualisation commune. Il a été choisi par FIPA comme un standard d'échange d'ontologies.

http://www.ai.sri.com/~okbc/

10.1.2. Les langages de représentation d'ontologies basés web

- **XML** (eXtensible Murkup Language): il s'agit d'un standard ouvert proposé par le W3C pour décrire les données et structurer les documents sur le web. XML tout seul ne peut être considéré comme un langage de représentation d'ontologie, alors que XML Schéma qui définit la structure, les contraintes, et la sémantique des documents XML peut en certain sens être utilisé pour spécifier des ontologies.
- **XOL** (XML Ontology Language): Il combine les frames et la syntaxe de XML pour représenter les classes, les taxonomies et les règles binaires, son apport est de fournir un format d'échange des définitions d'ontologies à travers un ensemble d'utilisateurs. Le développement de XOL est inspiré du language Ontolingua que lui diffère par la syntaxe basée XML au temps où le language Ontolingua est basé Lisp.
- **SHOE** (Simple HTML Ontology Extensions): Créé en 1996. Il s'agit d'une extension simple du langage HTML pour incorporer de la sémantique aux documents web traditionnels à travers l'annotation des pages HTML par des ontologies. SHOE combine les frames et les règles de production pour représenter les classes, les taxonomies, les relations, en plus des règles sous la forme de clauses de Horn qui vont

être utilisés par un moteur d'inférence pour dériver de nouvelles connaissances et il ne possède aucune ontologies, catégories, relations, ou inférences prédéfinies.

- **RDF** (Resource Description Framework): C'est un modèle de données décrivant les ressources web à travers un ensemble de nœuds représentant les objets ou les ressources, et un ensemble d'arcs représentant les propriétés formant ainsi un graphe de relations entre les différentes ressources, où chaque ressource se voit associée un identifiant unique, URI (Unique Resource Identifier), ces ressources sont donc liées entre elles ou à un littéral par le biais d'un triplet, « sujet, prédicat, objet », mais comme RDF seul ne définit aucune primitive pour créer des ontologies, il donne une base pour plusieurs langages de définition d'ontologie tel que RDFS.
- **RDFS:** Il ajoute une dimension sémantique aux graphes RDF notamment en permettant de définir des classes qui pourront être hiérarchisées offrant ainsi un ensemble de primitives tel que: rdfs: Class, rdf: Property, rdfs: SubClass, il permet aussi de définir des classes de classes, des classes de propriétés, des classes de littéraux (strings, entiers, booléens, etc.), et des classes de faits. Il offre en plus la possibilité de définir des instances de relations entre ressources et classes, des relations de subsomptions entre classes et relations de subsomptions entre propriétés en utilisant respectivement les propriétés de rdfs:
rdf: type, rdfs: SubClassOf, et rdfs: SubPropertyOf, et en utilisant les propriétés: rdfs: domain et rdfs: range, rdfs permet de restreindre ou de limiter la porté ou le domaine d'application de certaines relations. rdfs n'est pas considéré comme un vrai langage de représentation d'ontologie grâce à sa faiblesse d'expressivité tel qu'il ne permet pas de représenter plusieurs choses tel que par exemple l'intersection, l'union, le complément de classe, les cardinalités, les propriétés de relations tel que la transitivité, la symétrie, l'inverse, etc.
- **OIL** (Ontology Inference Layer): Compatible avec RDF(S), permettant ainsi de décrire les ontologies en combinant les primitives de modélisation utilisées dans les langages de frames et le raisonnement formel des logiques de description pour exprimer les ontologies sur le web. Il est de faible expressivité vu qu'il ne supporte pas tout les types tels que les types concrets, d'où la volonté de pouvoir conserver un test de subsomption décidable.

- **DAML+OIL:** Il s'agit d'une extension RDF(S), en ajoutant des contraintes aux valeurs attribués aux propriétés et en spécifiant les propriétés qu'une classe peut avoir. Il supporte les types primitifs existant dans XML ainsi que la définition d'un certain nombre d'axiomes tel que l'équivalence de classes ou de propriétés. Il est équivalent à une logique descriptive très expressive, mais demeure assez limité dans le contexte d'ontologies d'application sur le web.
- **OWL** (Ontology Web Language): Comme son nom l'indique OWL est un langage de représentation d'ontologies destinées à être publiées et partagées sur le web. Il s'agit de la dernière recommandation du fameux W3C dans ce sens et constitue une surcouche de RDF(S) qui est lui-même une surcouche de XML. Voir la figure 1.5.

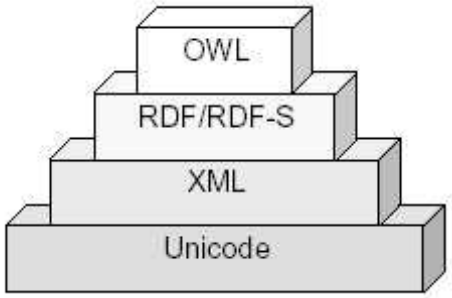


Figure 1-4. Relation entre Unicode, XML, RDF, et OWL *(Aubry, 2007)*

Il s'agit donc d'un langage qui permet d'hiérarchiser les classes représentant les classes d'un domaine particulier à travers des relations de types « is-a », tout en spécifiant leurs propriétés qui peuvent être héritées à partir d'autres classes parents dans l'arborescence, ainsi que des contraintes de construction du graphe ou de la structure hiérarchique tel que les cardinalités minimales et maximales des relations qui peuvent être, autres que les relations de type « is-a », des relations logiques tel que l'(in)égalité, l'union, l'intersection ou la disjonction entre classes. Une ontologie OWL est caractérisée par un entête d'axiomes et de faits représentant les métadonnées décrivant l'ontologie, où les axiomes décrivent les classes ou les relations, tels que les labels fournissant les noms des classes ou des relations. En plus des commentaires pouvant être exploités pour générer une documentation associée à l'ontologie en cours de développement. Pour répondre aux besoins d'expressivité ontologique croissante OWL s'est évolué en mettant en œuvre trois sous langages en l'occurrence OWL-Lite, OWL-DL, et OWL-Full:

OWL-Lite: La forme la plus simplifiée (Classification hiérarchique) et la plus facile à implémenter (contraintes simples) d'OWL. Il s'agit d'une extension de RDF(S), et pour pouvoir exprimer les notions citées ci-dessus, OWL-Lite utilise en plus des constructeurs de RDF(S): Class, Property, subClass, SubProperty, domain, range, etc. De nouveaux constructeurs tel que IntersectionOf, equivalentClass, equivalentProperty, Cardinality, MinCardinality, MaxCardinality, InverseOf, DataTypeProperty, ObjectTypeProperty (Type de données), AllValuesFrom, SomeValuesFrom (restriction sur les rôles), Ontology, Imports (informations d'entête),etc.

OWL-DL: Il s'agit d'une extension de OWL-Lite (en ajoutant des axiomes et des combinaisons binaires des expressions de classes et de relations), principalement basée sur la logique de description offrant ainsi une expressivité plus importante que celle d'OWL-Lite tout en assurant la complétude (tous les objectifs et buts démontrables peuvent être déduits), et de la décidabilité (tout raisonnement doit se terminer à un temps fini), mais il présente certaines restrictions à l'utilisateur par exemple une classe ne peut pas être une instance d'une autre classe.

OWL-Full: il s'agit d'une extension d'OWL-DL. Il comble les lacunes des deux premiers sous langages OWL, en accroissant ainsi l'expressivité ontologique et en n'imposant aucune restriction à l'utilisation, mais aucune garantie de calculabilité de l'ontologie ainsi créée n'est pas fournie.

Donc en fonction de leurs besoins, les développeurs d'ontologies choisissent le sous langage qui leur convient le plus, qu'il s'agisse de la simplicité et de la facilité d'implémentation (OWL-Lite), de la richesse d'expression (OWL-DL), ou d'éviter les restrictions pouvant leur y imposées (OWL-Full).

Il est à noter qu'une ontologie OWL-Lite valide est une ontologie OWL-DL et OWL-Full valide.

- ***Critères de sélection d'un langage de représentation d'ontologies:*** Selon (Saïd, 2006), les critères de sélection du langage de représentation d'ontologie les plus souvent retenues sont: L'expressivité reflétant la quantité de connaissances ou le nombre de classes pouvant être utilisés pour décrire les composants d'une ontologie, et la performance qui définit la capacité et la facilité de représentation de connaissances du domaine. Il est à noter que plus le langage est expressif moins il sera performant. Il y a

aussi un autre dilemme entre l'expressivité et le raisonnement, tel que parfois l'expressivité du langage doit être limitée pour assurer un bon service de raisonnement.

10.2. Les éditeurs d'ingénierie ontologique:

Les éditeurs jouent un rôle primordial en assistant l'ontologiste à construire et à développer les ontologies en suivant une méthodologie de conception plus ou moins complète et en respectant les principes du cycle de vie, et en exploitant des mécanismes de visualisation et de vérification du modèle conceptuel résultant des vérificateurs logiques pour vérifier la satisfiabilité du modèle spécifié. Dans ce qui suit nous allons présenter un certain nombre d'éditeurs et d'outils environnementaux de création et de gestion d'ontologies, nous avons donc choisi les outils les plus utilisés à savoir:

10.2.1. OntoEdit (Sure et al., 2002)**:** (Ontology Editor): Développé à l'université de Karlsruhe, il est indépendant de tout langage de représentation, et plusieurs plugins peuvent être intégrés permettant la visualisation graphique de l'ontologie, le test de la cohérence de l'ontologie, etc. Tels que ONTOKICK pour générer les spécifications de l'ontologie et OntoBroker pour inférer de nouvelles connaissances, et aussi pour importer et exporter l'ontologie vers de multiples langages tels que RDFS, DAML+OIL, et FLogic.

10.2.2. Protégé 2000 (Horridge et al., 2004)**:** C'est un environnement graphique modulaire d'ingénierie ontologique développé au sein de l'SMI de Standford. Il est composé d'un éditeur et d'une librairie de plugins permettant l'édition, la visualisation, le contrôle ainsi que l'extraction d'ontologies à partir de sources textuelles. Une ontologie sous protégé 2000 est constituée de plusieurs hiérarchies de classes, où chaque classe est décrite par un ensemble d'attributs (slots) pouvant être spécifié en remplissant les formulaires fournis par protégé. Le succès de protégé 2000 est justifié par la conception ainsi que l'architecture logiciel très bien spécifiée de son interface en plus des nouvelles fonctionnalités apportées par les plugins pouvant être intégrés dedans, tel que le plugin Prompt pour la fusion de plusieurs ontologies.

10.2.3. Web ODE (Arpírez et al., 2001**):** Développé au sein du même laboratoire où ODE a été développé. Il s'agit d'une plate forme de conception d'ontologies

fonctionnant en ligne favorisant ainsi le travail collaboratif, il s'agit donc d'une adaptation d'ODE pour le web.

10.2.4. WebOnto (web ontology) (Domingue, 1998): Développé à l'Open University, et supporte OCML comme un langage de représentation.

10.2.5. OilEd (OIL Editor) (Bechhofer et al., 2001): Développé à l'université de Manchister. Il favorise la construction d'ontologies de petites et de moyennes tailles formalisées en utilisant les standard DAML+OIL ainsi que la logique de description. Il offre, grâce à son moteur d'inférence des services de raisonnement permettant de tester la satisfiabilité et de découvrir des subsomptions restées implicites dans l'ontologie. Il permet l'export d'ontologie sous les formats RDF, DAML+OIL, et OWL.

11. La médiation d'ontologies.

Elle constitue une nouvelle voie de recherche qui reste toujours très riche et qui étudie les techniques permettant la réutilisation d'ontologies tout en identifiant les ressemblances ainsi que les différences entre eux. Ces différences constituent un vrai obstacle en face de l'interopérabilité entre ces ontologies mais elles peuvent heureusement être réconciliées à travers la dite « médiation d'ontologies ». Selon (De Bruijn et al., 2006), la médiation d'ontologie peut être faite de différentes manières à savoir: la fusion d'ontologies visant à créer une nouvelle et unique ontologie à partir de plusieurs ontologies, l'alignement d'ontologies visant à découvrir (semi) automatiquement les correspondances entre les ontologies et le mapping d'ontologies visant à représenter les correspondances entre ces ontologies. Dans ce qui suit nous détaillons ces trois techniques de médiation d'ontologies:

11.1. L'Alignement d'ontologies

Il s'agit de la spécification des rapports entre deux ou plusieurs ontologies sources à travers un processus de découverte de correspondances. En d'autres termes c'est le processus qui consiste à chercher des rapports entre les entités des ontologies en question, sans modifier ces ontologies. Les rapports ainsi détectés sont appelés « Les mappings ». Les entités des ontologies peuvent être les classes, les relations, les fonctions, les instances, les axiomes, les règles, etc. Les algorithmes d'alignement sont classifiés selon deux dimensions:

- Selon la première dimension on distingue l'alignement basée schéma et l'alignement basée instance:
  - Dans l'alignement basé schéma, différents aspects de classes et de relations des ontologies sont pris en compte pour identifier les similarités en se basant sur des mesures particulières.
  - Dans l'alignement basé instance, les instances des différentes classes des différentes ontologies sont comparées entre elles pour dégager les similarités entre les classes .
- Selon la deuxième dimension on distingue: L'alignement au niveau élément et l'alignement au niveau structure:
  - L'alignement au niveau élément découvre les similarités en comparant les propriétés de certaines classes et relations (tel que nom).
  - L'alignement au niveau structure découvre les similarités en comparant les structures des ontologies.

Il est à noter que les derniers types d'alignement peuvent être combinés pour aboutir à de meilleurs résultats.

11.2. Les mappings entre ontologies

C'est une spécification déclarative du sens commun entre deux ontologies à travers la représentation des correspondances entre elles. Ces correspondances sont enregistrées isolément des ontologies constituant ainsi une partie à part des ontologies sources et représentant les liens entre les entités correspondantes à travers des axiomes spécifiés dans un langage de mapping spécifique. Ces correspondances peuvent être de différents types, en l'occurrence, l'équivalence, la subsomption, l'exclusion ou l'incompatibilité. Les mappings entre ontologies sources sont le résultat du processus d'alignement. Ils peuvent être exploités pour exécuter différents autres processus émergeant dans le Web sémantique tel que la fusion et le versionning d'ontologies, la gestion de connaissances, l'intégration sémantique, etc. Ceci peut alors servir l'interrogation de bases de connaissances hétérogènes à travers une interface utilisateur commune ainsi que la transformation ou l'interprétation des données d'un formalisme de représentation vers un autre.

11.3. La fusion d'ontologies

Il s'agit de créer une nouvelle et unique ontologie qui représente l'union des deux ontologies sources de façon à regrouper toutes les connaissances contenues dans les deux ontologies, en d'autres termes toutes les ressemblances et les dissemblances présentées par les deux ontologies doivent être reflétées par l'ontologie créée de ce processus.

12. Autres opérations de réutilisation d'ontologies

Une multitude d'opérations de réutilisation d'ontologies est collectée par Gomez-Perez. Dans la section suivante nous en présentons un résumé:

- **Alignement d'ontologies** (Ontology Aligning): Il s'agit de trouver les correspondances entre deux (ou plusieurs) ontologies puis les sauvegarder et/ou les exploiter dans un certain contexte.
- **Annotation d'ontologies:** IL s'agit d'enrichir l'ontologie avec des informations supplémentaires sous forme de méta données ou de commentaires.
- **Comparaison d'ontologies:** IL s'agit de trouver les différences entre deux (ou plusieurs) ontologies ou entre deux (ou plusieurs) modules d'ontologies.
- **Adaptation d'ontologie:** Comme son nom l'indique, il s'agit de faire adapter une ontologie aux besoins d'un utilisateur spécifique.
- **Enrichissement d'ontologie:** Il s'agit de faire étendre ou prolonger une ontologie en ajoutant de nouvelles structures conceptuelles (tels que les classes, les relations, les rôles, etc.)
- **Évolution d'ontologie:** IL s'agit de faciliter la modification d'une ontologie tout en préservant sa consistance, l'évolution peut être vue comme une séquence de différentes activités pendant le développement de l'ontologie.
- **Extension de l'ontologie:** IL s'agit d'une activité d'enrichissement afin d'étendre la largeur d'une ontologie.
- **Intégration d'ontologie:** Comme son nom l'indique, il s'agit d'ancrer une ontologie dans une autre.
- **Mapping d'ontologies:** IL s'agit de trouver les correspondances entre deux (ou plusieurs) ontologies puis les sauvegarder et/ou les exploiter dans un certain contexte.

- **Matching d'ontologies:** IL s'agit de trouver ou de découvrir les relations ou les liens de correspondances entre les entités ou les modules de différentes ontologies.
- **Fusion d'ontologies:** IL s'agit de créer ou de construire une nouvelle ontologie ou un nouveau module d'ontologie à partir de deux ou plusieurs ontologies ou modules d'ontologies sources.
- **Modification d'ontologie:** IL s'agit de changer ou modifier une ontologie sans prendre en charge la préservation de la consistance.
- **Modularisation d'ontologie:** IL s'agit d'identifier un ou plusieurs modules dans une ontologie dans le but de supporter ou de faciliter la réutilisation ou la maintenance.
- **Extraction de module d'ontologie:** IL s'agit d'obtenir des modules concrets à partir d'ontologie pour être utilisé dans une proposition particulière (pour obtenir par exemple un sous vocabulaire particulier de l'ontologie originale).
- **Partitionnement d'ontologie:** IL s'agit de diviser ou de partitionner une ontologie en un ensemble (pas nécessairement disjoint) de modules qui peuvent être traités séparément et dont l'union forme l'ontologie originale.
- **Réingénierie d'ontologie:** IL s'agit d'un processus de rétablissement et de transformation d'un modèle conceptuel modélisant une ontologie déjà fini et implémenté vers un modèle conceptuel plus correct et plus complet qui va être ré-implémenté.
- **Restructuration d'ontologie:** IL s'agit de corriger et de réorganiser les connaissances contenues dans un modèle conceptuel initial, et de détecter les connaissances du domaine qui manque dans le modèle conceptuel.
- **Réparation d'ontologie:** Il s'agit d'un processus lancé par une ontologie spécifique qui s'appelle « diagnosis ontology », et qui consiste à corriger et à résoudre les erreurs tels que l'incomplétude ou l'incohérence.
- **Réutilisation d'ontologie:** Il s'agit d'utiliser une ontologie ou un module d'ontologie pour résoudre un certain problème, en exploitant une ontologie ou un module d'ontologie.
- **Recherche d'ontologie:** IL s'agit de trouver les ontologies ou les modules d'ontologie candidats pour être réutilisé.

õ **Résumé d'ontologie:** IL s'agit d'obtenir une abstraction ou un résumé du contenu de l'ontologie.

õ **Translation d'ontologie:** IL s'agit de changer le formalisme ou le langage de représentation de l'ontologie vers un autre. La translation d'ontologie peut faire partie du processus de ré-ingénierie d'ontologie.

õ **La mise à jour de l'ontologie** (ontology update): Il s'agit du moindre changement effectué sur une ontologie sans faire évoluer la version de l'ontologie.

õ **Révolution de l'ontologie** (ontology upgrade): Il s'agit de remplacer une ontologie (existante) par une nouvelle version.

õ **Validation d'ontologie:** Il s'agit de l'évaluation de l'ontologie tout en comparant le sens des définitions de l'ontologie avec le modèle visant à conceptualiser le domaine du problème (en répondant à la question: est ce qu'on a développé la bonne ontologie ?)

õ **Ontology versionning:** IL s'agit de porter les changements d'ontologies tout en créant diverses versions de l'ontologie.

13. Conclusion

Avant l'émergence du concept ontologique, les connaissances étaient principalement stockées dans des Bases de Données. Celles-ci ont connu beaucoup de problèmes et de difficultés de partage et de réutilisation d'informations et de connaissances stockées dedans. Pour confronter ces formes de problèmes et d'autres, les ontologies sont exploitées comme étant des modèles de représentation et de stockages très prometteuses qui supportent le partage et la réutilisation des connaissances et des informations. En effet, et vu leur potentiel de fournir des sémantiques explicites aux connaissances et informations, les ontologies sont largement exploitées comme étant des moyens pour résoudre les problèmes d'hétérogénéités des informations sur le web. Dans ce sens, les ontologies ont alors joué un rôle fondamental pour décrire les sémantiques de données pour le web sémantique, tout en représentant les spécifications formelles des concepts des domaines ainsi que les relations entre eux. Pour qu'une ontologie puisse atteindre son objectif, et pour que son application porteuse puisse éviter des problèmes critiques et catastrophiques, on doit s'assurer que

l’ontologie exploitée est libérée des différents types d’erreurs et anomalies. Pour ce faire, l’évaluation du contenu ontologique est une phase à ne pas éviter avant toute exploitation et/ou intégration d’une ontologie dans une application et/ou système d’informations. Dans ce chapitre, nous avons présenté un état de l’art sur les ontologies, où nous avons concentré sur les aspects de conception et de créations des ontologies. D’abord, nous avons définie le concept ontologique, ses composants, et ses critères de classification, en plus de ses principes de bases de construction et d’utilisation, guidant de façon primordiale le processus d’ingénierie ontologique permettant aux développeurs d’applications porteuses et aux systèmes d’informations de partager un référentiel commun, servant de base à la collaboration et à l’interopérabilité entre plusieurs systèmes et/ou individus d’une communauté au sein d’un domaine particulier. Ensuite, nous avons présenté une classification des erreurs de conception et de construction d’ontologies en plus d’une description plus ou moins détaillée des différents langages de représentation, méthodologies de construction et éditeurs de développement d’ontologies. En fin, nous avons discuté le problème de la réutilisation d’ontologies pour introduire le chapitre de réutilisation d’ontologies à travers la fusion de deux ou plusieurs ontologies.

Chapitre 2 . Fusion d'ontologies – État de l'art -

1. Introduction

L'interopérabilité est une caractéristique cruciale dans de nombreux domaines, notamment le web sémantique. Elle peut se définir comme « la capacité de fournir des applications capables de fonctionner conjointement sur les mêmes données indépendamment de la localisation géographique de ces données et des applications. »[1]. On peut parler d'interopérabilité à plusieurs niveaux. Au niveau sémantique elle est nécessaire pour résoudre le problème d'hétérogénéité entre ontologies porteuses de sémantiques communes de différentes provenances afin de concilier entre ces sources d'informations. Cette interopérabilité est alors très recommandée pour combiner des ontologies hétérogènes et distribuées (Choi, Song, & Han, 2006). Elle permet aux chercheurs et aux agents software de travailler dans un environnement plus souple et favorisant plus de collaboration (Saleem, 2006). Dans la littérature, deux techniques majeures d'interopérabilité existent, en l'occurrence : l'alignement et la fusion d'ontologies. Lors du processus d'alignement les ontologies sources persistent tout en établissant des liens sémantiques entre elles, ce qui permet à l'une d'utiliser les connaissances de l'autre. Les liens sémantiques existant entre les ontologies alignées sont appelés « Les mappings ». L'alignement est généralement adopté quand les ontologies sources couvrent des domaines d'application complémentaires. Alors que lors du processus de fusion, les ontologies sources vont être remplacées par une seule et nouvelle ontologie cohérente qui représente l'union des ontologies sources. Ce processus est généralement adopté quand les ontologies sources couvrent des domaines similaires ou connexes. Nous avons présenté en détail, ces techniques d'interopérabilité entre les ontologies dans des travaux précédents, à savoir (Amrouch et al., 2012a) et (Amrouch et al., 2012b).

L'objectif de ce chapitre est de présenter un état de l'art sur la fusion d'ontologies. Nous y présentons en particulier le processus de fusion, ses techniques et ses étapes, les différents types de problèmes auxquels il peut être confronté. Nous détaillons

[1]Dictionnaire de l'informatique et d'internet : http://www.dicofr.com/cgi-bin/n.pl/dicofr/definition/20010101002651

également la notion de similarité qui est à la base de l'identification des classes similaires qui doivent être fusionnées. A ce stade, nous décrivons le processus de découverte de correspondances et les notions inhérentes. A la fin, nous présentons quelques applications de la technique de fusion d'ontologies ainsi qu'un survol des principaux travaux de fusion, automatique et/ou semi automatique qui existent dans la littérature et nous terminons par une conclusion.

2. Le processus de fusion d'ontologies

Selon (Robin & Uma, 2010), un processus de fusion d'ontologies est semblable à un processus de construction d'une nouvelle ontologie. D'une part, et selon (Gomez-Perez, Fernández-López, & Corcho, 2006), construire une ontologie revient à définir ses classes, les arranger dans une hiérarchie taxonomique, décrire les valeurs attribuées aux slots et fixer les valeurs de slots des instances. D'autre part, et selon (Ghidini & Giunchiglia, 2003), la fusion d'ontologies est vue comme étant le processus de création d'une nouvelle et unique ontologie cohérente à partir de deux ou plus ontologies sources préexistantes et reliées au même domaine d'application. La nouvelle ontologie va alors remplacer les ontologies sources.

En revanche, pour créer une seule ontologie cohérente à partir de plusieurs ontologies sources, un processus de fusion d'ontologies doit tout d'abord résoudre différents types d'hétérogénéité entre ontologies sources. Parmi ceux les plus connus : (i) Les hétérogénéités au niveau langage ou hétérogénéités syntaxiques, causé par l'utilisation de différents langages de représentation d'ontologies. (ii) Les hétérogénéités au niveau ontologies ou hétérogénéités sémantiques tels que les synonymes, les homonymes, les hyponymes, etc. Ces différentes formes d'hétérogénéités doivent alors être identifiées et résolues lors de l'étape d'identification de similarités. Ce processus est très coûteux, long, difficile et sujet à l'erreur s'il est effectué manuellement, surtout dans le cas des systèmes volumineux et complexes.

2.1. Définition

Dans (Sowa, 1997), l'auteur définit la fusion d'ontologies comme étant le processus de trouver des points communs entre deux ontologies différentes A et B et de dériver une nouvelle ontologie C facilitant l'interopérabilité entre des systèmes automatiques qui sont basés sur les deux ontologies A et B. L'ontologie C peut donc remplacer les deux

ontologies A et B ou jouer le rôle d'un intermédiaire entre deux systèmes : l'un basé sur l'ontologie A et l'autre basé sur l'ontologie B.

Formellement, la fusion d'ontologies peut être définie par la fonction « *f* », qui à partir de deux (ou plus) ontologies sources O_1 et O_2, une mesure de similarité « *S* », un ensemble de seuils critiques « *c* » et un ensemble de ressources lexicales «*R* », produit une nouvelle et unique ontologie « O_m » regroupant toutes les ressemblances et dissemblances présentées par les otologies sources O_1 et O_2. La fonction *f* peut être illustrée par la figure 2.1 :

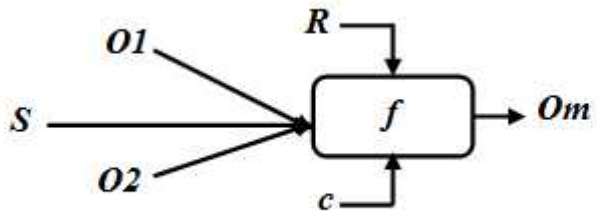


Figure 2-1. Illustration formelle de la fonction de fusion d'ontologies

2.2. Les étapes d'un processus de fusion d'ontologies

Dans ce qui suit, nous présentons une description des étapes communes qui sont adoptées par la plupart des outils et méthodes de fusion d'ontologies, qui existent dans la littérature. Nous allons illustrer ces étapes par la figure 2.2 :

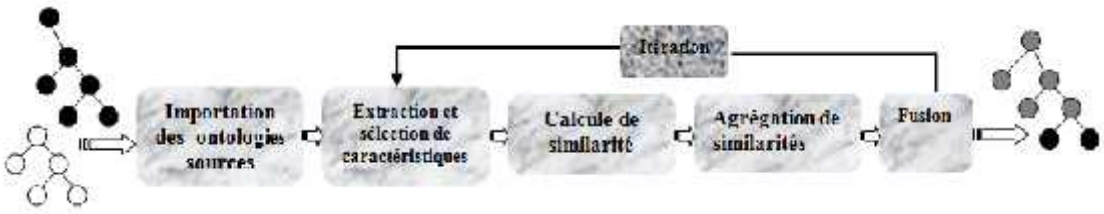


Figure 2-2. Les étapes génériques d'un processus de fusion d'ontologies

i. **Importation des ontologies sources :** Pour qu'elles puissent être fusionnées, les ontologies sources doivent être spécifiées dans un même langage de représentation.

ii. **Extraction et sélection de caractéristiques :** Lors de cette étape, les caractéristiques des ontologies sources qui vont passer par un processus de mesure de similarité sont sélectionnées et extraites. Ces caractéristiques peuvent être de différents types d'entités ontologiques, tels que les classes (ou les concepts), les propriétés (propriété de type de données et propriété d'objet), les instances, etc. Ensuite, et selon

la technique de fusion adoptée (top-down, middle-out ou bottom-up) les paires de classes à comparer sont identifiées à chaque itération, tel que la première classe appartient à la première ontologie, et le seconde appartient à la deuxième. Selon les astuces d'optimisation adoptées, toutes les paires d'entités ontologiques ou seulement un sous ensemble de ces paires seront considérées et comparées. Ainsi, dans les paires retenues, chaque classe de la première ontologie est comparée avec toutes les classes de la deuxième ontologie et les comparaisons jugées non nécessaires sont omises.

iii. **Calcul de similarité :** Lors de cette étape, la similarité entre les pairs de classes précédemment sélectionnées est calculée. Plusieurs mesures de similarité lexicales, syntaxiques et sémantiques peuvent être utilisées par la même méthode.

➢ Notion de similarité

Deux (ou plusieurs) objets sont dits similaires s'ils partagent un ensemble important de caractéristiques ou d'attributs. Ils sont alors dit sémantiquement similaires s'ils partagent un ensemble de sens. En effet, la notion de similarité constitue un support implicite mais primordial dans de nombreuses applications surtout celles qui font appel à la recherche d'information.

Dans certains contextes, il est très utile de déterminer la similarité entre deux ontologies (déterminer si deux ontologies sont proches ou non, ou encore chercher l'ontologie la plus proche d'une autre), à titre d'exemple :

- Dans les moteurs de recherche sémantiques qui retournent des ontologies correspondantes à une requête, il serait utile d'introduire le bouton « Find Similar Ontologies » (D' Aquin et al., 2007). Cela peut aussi être utilisé dans l'ordre des réponses à une requête (Ontology Ranking) en fonction de la proximité des ontologies (Alani & Brewster, 2005).
- Dans le contexte de la fusion d'ontologies, la notion de similarité et surtout la similarité sémantique est un concept primordial lors de l'identification des classes similaires qui doivent être fusionnées en une seule classe dans l'ontologie résultante.
- …

Une mesure de similarité particulière ne peut satisfaire toutes sortes d'applications. Mais, en fonction de la situation en cours, certaines mesures sont plus appropriées.

iv. **Agrégation de similarité:** Lors de cette étape, toutes les similarités calculées sont agrégées (ou combinées) en une seule formule, selon la technique d'agrégation adoptée (séquentielle, parallèle ou mixte).

v. **Fusion:** Lors de cette étape, et en se basant sur la valeur de similarité précédemment calculée, les classes jugées similaires sont fusionnées en une seule classe et les classes jugées dissimilaires sont directement copiées dans l'ontologie fusionnée.

vi. **Itération:** Ce processus est réitéré pour chaque paire de classes jusqu'à la construction complète de l'ontologie fusionnée qui doit refléter toutes les ressemblances et dissemblances représentées dans les ontologies sources.

2.3. Caractéristiques d'un processus de fusion d'ontologies

2.3.1 Les ontologies sources (O1, O2)

Plusieurs types de contraintes peuvent être appliqués aux ontologies sources, tel que la contrainte de cardinalité informant sur le nombre d'ontologies à fusionner, deux (ou plusieurs) ontologies. Le langage de représentation des ontologies à fusionner, l'hétérogénéité des ontologies sources, est-ce qu'elles sont représentées par le même langage de représentation ou par des langages de représentation différents, etc.

2.3.2 Le type d'entité d'entrée

Un processus de fusion d'ontologies peut s'exécuter sur différents types d'entités ontologiques, tel que les classes, les propriétés, les instances, etc.

2.3.3 Les paramètres (R, S, C)

Dans un processus de fusion d'ontologies, trois paramètres importants doivent être identifiés, en l'occurrence, (R) les ressources lexicales tel que wordnet, (S) les mesures de similarités lexicales, sémantiques ou encore apprentissage automatique et (C) valeurs de seuils critiques.

2.3.4 Spécificités du sous processus de découverte de correspondances

Le processus de découverte de correspondances entre classes similaires peut être soit :

- Purement manuel, exécuté par l'ontologiste,

- semi-automatique, assisté par ordinateur, l'ontologiste peut intervenir soit pour valider les résultats obtenus de manière automatique, choisir la fonction ou la mesure de similarité, ou choisir les correspondances appropriées parmi un ensemble de correspondances proposées.
- Purement automatique ne demandant aucune intervention humaine.

2.3.5 L'opérationnalisation des correspondances découvertes

Les classes appartenant aux différentes ontologies sources doivent finalement être soumises à un processus de fusion pour créer une nouvelle et unique ontologie rassemblant toutes les connaissances des ontologies sources**.**

2.4. Problèmes liés à la fusion d'ontologies

Selon (Klein, 2001) et (Predoiu et al., 2005), la fusion d'ontologies peut souffrir de plusieurs types de problèmes dont le mismatch constitue le problème le plus commun. Un mismatch indique une mauvaise correspondance entre des entités des ontologies sources. La Figure 2.3 résume les problèmes rencontrés dans un processus de fusion d'ontologies. Ils peuvent être de deux types principaux : les problèmes d'ordre technique et les problèmes d'ordre pratique

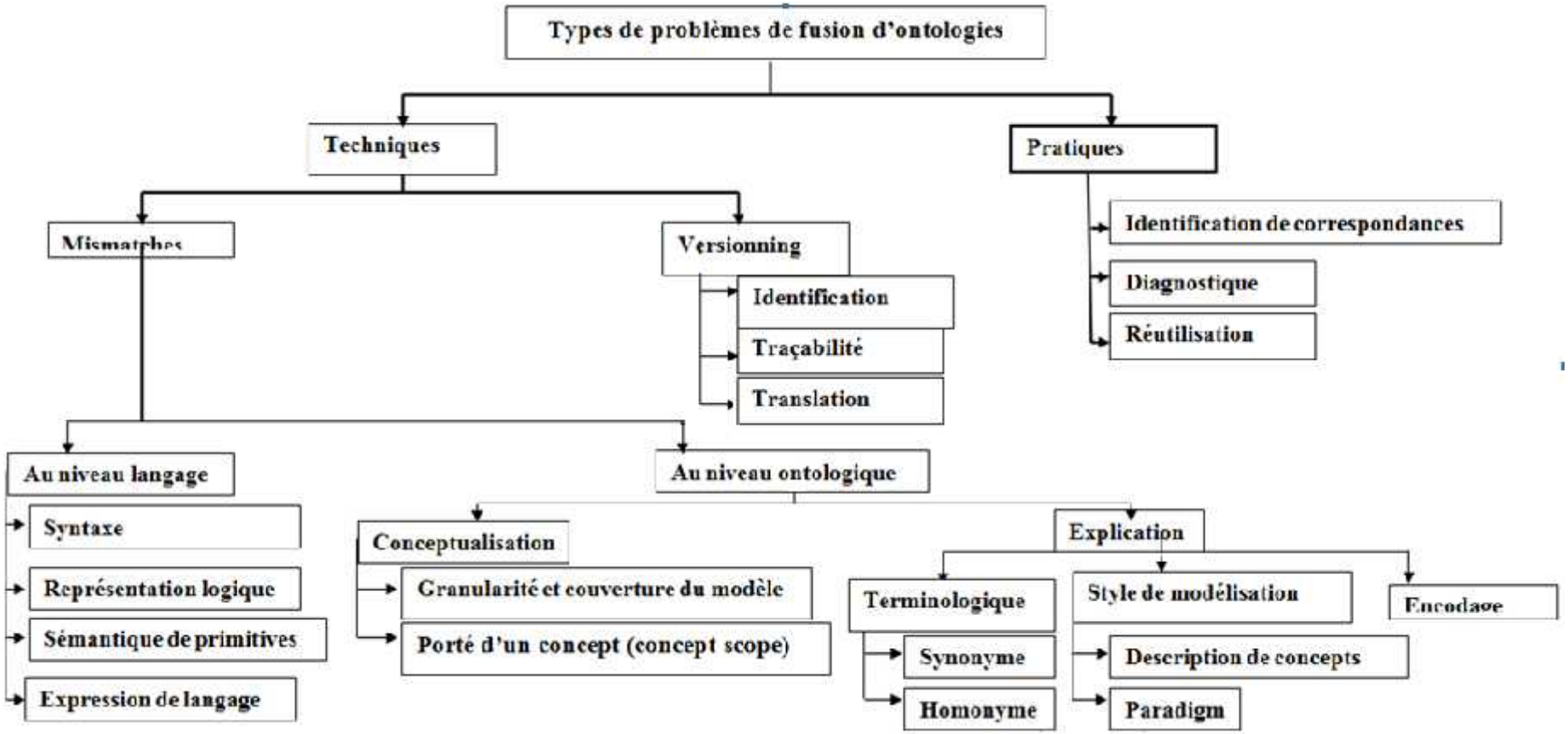


Figure 2-3. Les différents types de problèmes de fusion d'ontologies

2.4.1. Les problèmes techniques

2.4.1.1. Mismatches entre ontologies

Plusieurs types de mismatches entre ontologies sont constatés dans la littérature des ontologies et sont classifiés comme suit:

a. Mismatches au niveau langage: Peuvent apparaître lorsque les ontologies sources sont écrites dans des langages de représentation différents. Ils décrivent de plus les écarts entre les mécanismes décrivant les classes et les relations et reflètent des différences non sémantiques entre les ontologies sources. Ils peuvent eux-mêmes être de différents niveaux:

- Syntaxe: Il est évident que différents langages de représentation d'ontologies utilisent différentes syntaxes. Par exemple, pour définir la classe *« chaise »* en RDF, on écrit: *<rdfs:Class ID = « chaise »>*. Dans LOOM, il est décrit par: *(defclasse chaise).*
- Représentation logique: apparaît lorsque des représentations logiques qui sont syntaxiquement différentes mais logiquement équivalentes sont utilisées pour représenter le même objet, tel que l'expression de la disjonction peut être faite par *(disjoint A B)* mais aussi par *(A subclass-of (not B), B subclass of (not A)).*
- Sémantique de primitives: apparaît lorsque des expressions sont syntaxiquement équivalentes mais sémantiquement différentes (de différents langages de représentation). En d'autres termes, même si les deux ontologies utilisent la même syntaxe, leurs sémantiques peuvent être différentes. Par exemple, la syntaxe OIL-RDFSchéma interprète: *<rdfs:domain>* comme l'intersection des arguments, alors que RDFSchéma l'interprète comme l'union des sémantiques des arguments.
- Expressivité du langage: il s'agit de la différence d'expressivité entre deux langages qui implique qu'un objet est exprimable dans un langage et non dans un autre. A titre d'exemple : certains langages détiennent des constructeurs pour exprimer la négation, supporter la notion d'ensemble, de valeurs par défauts, etc. alors que d'autres non.

➢ Tous ces types de problèmes, peuvent être résolus par la translation de l'une des deux ontologies sources vers le langage de représentation et/ou le formalisme de représentation de l'autre ontologie.

b. Mismatches au niveau ontologique: peuvent apparaître lorsque les ontologies sont écrites dans le même langage de représentation, mais modélisées différemment, ce qui se traduit par des différences sémantiques entre les ontologies sources. Ce type de mismatch admet lui aussi plusieurs types:

• au niveau de la conceptualisation: décrivant la différence dans la conceptualisation ou l'interprétation du domaine donnant lieu à différentes classes ontologiques et/ou différentes relations entre elles:

- Granularité et couverture du modèle: décrivant la différence dans la partie du domaine couvert par l'ontologie et/ou le niveau de détail par lequel le domaine est modélisé.

- Porté d'une classe (class scope) : décrivant la différence de l'extension de la classe, i.e. différence dans l'ensemble des instances. Lorsque deux classes semblent représenter la même classe, mais n'ont pas exactement les mêmes instances (class mismatch).

Le problème de différence de conceptualisation ne peut être résolu automatiquement mais nécessite la décision d'un expert du domaine.

• Au niveau explicatif: décrivant la différence dans la spécification de la conceptualisation engendrant ainsi des différences dans les définitions des termes, et leurs combinaisons. Une classification des types de différences au niveau explication est comme suit:

- Selon le style de la modélisation:

Paradigme de représentation: Différents paradigmes peuvent être utilisés pour représenter les classes. Par exemple un modèle représentant la classe « temps » par des intervalles de temps, alors qu'un autre la représente par des points.

Description de classes (convention de modélisation): plusieurs choix peuvent être pris pour modéliser les classes dans une ontologie. Par exemple, la distinction entre deux classes peut être faite soit par l'utilisation des attributs de qualification soit par l'utilisation des séparateurs de classes. Un autre choix de description de classes est celui basé sur les hiérarchies « is-a », où la distinction des classes (leurs attributs) peut être faite plus haut ou plus bas dans la hiérarchie. Par exemple, dans l'assertion

suivante, pour différencier entre publication scientifique et non scientifique, une dissertation peut être modélisée comme suit:

Dissertation: <*book*<*scientific publication*<*publication*, ou encore

Dissertation: <*scientific book*<*book*<*publication*

- Mismatch terminologique: sont représentés par :

Synonymes (term mismatchs): Deux termes sont synonymes s'ils sont sémantiquement équivalents mais représentés par des noms différents. Pour résoudre ce problème, il est recommandé d'utiliser des thésaurus, ou des dictionnaires, mais il faut faire attention aux différences d'extensions possibles.

Homonymes (classe mismatchs): Ce problème apparaît lorsque des classes sémantiquement différentes portent le même nom.

• Encodage: Des valeurs dans différentes ontologies peuvent être codées par différents formats. Par exemple, la date peut être représentée par « dd/mm/yyyy » ou par « dd-mm-yyyy », une distance peut être représentée en kilomètres ou en miles.

2.4.1.2. Versioning d'ontologies

Le versioning est le processus de création de différentes versions pour la même ontologie. Lors d'un processus de versioning, les exigences suivantes sont imposées sur un schéma de versioning (avec difficultés croissantes):

a. **Identification:** Pour chaque utilisation d'une classe ou d'une relation, le cadre du versioning doit fournir une référence non ambiguë pour étendre une définition.

b. **Traçabilité:** Le cadre du versioning doit établir une relation entre une version d'une classe ou d'une relation et une autre version de cette classe ou de cette relation.

c. **Translation:** Le cadre du versioning doit établir automatiquement des translations (conversions) d'une version à une autre pour permettre un accès transparent à la dernière version.

2.4.2.Les problèmes pratiques

En plus des problèmes techniques, il existe des problèmes pratiques liés à la fusion d'ontologies. En pratiques, les grandes difficultés dans un processus de fusion sont principalement:

• **La découverte de correspondances** : Il est difficile d'identifier les termes qui vont être appariés ensemble.

- **Le Diagnostique:** Les conséquences ou les résultats d'un appariement spécifique sont difficiles à diagnostiquer.
- **La réutilisabilité:** Les ontologies sources utilisées pour la fusion continuent à évoluer, les correspondances crées pour la fusion doivent être le plus possible réutilisables pour fusionner les ontologies révisées.

3. Techniques de fusion d'ontologies:

Les techniques de fusion d'ontologies décrivent la façon de traiter les opérations de fusion. Elles servent de base dans la conception d'outils de fusion automatique ou semi-automatique. Trois techniques de fusion d'ontologies sont proposées dans la littérature : Bottom-up, top-down et middle-out. En fonction de la nature de l'application et des besoins spécifiques de la fusion une de ces trois techniques est choisie.

3.1. Technique « Bottom-Up »

Cette technique est proposée dans (Stumme & Maedche, 2001). Comme son nom l'indique, il s'agit d'une approche dans laquelle on remonte des classes les plus spécifiques vers les classes les plus générales, lors de l'identification des classes similaires, en appliquant plusieurs formes d'heuristiques, par exemple en comparant les noms et les définitions de deux classes et en contrôlant la proximité de deux classes dans la hiérarchie de classes.

3.2. Technique « Top-Down »

Dans cette approche, les classes les plus génériques sont définies en premier puis elles sont affinées ou spécialisées.

3.3. La technique « Middle-Out »

Il s'agit d'une approche où les classes les plus importantes sont définies en premier puis sont généralisées ou spécialisées, tout en gardant le contrôle du niveau de détail qui ne peut être contrôlé avec l'approche « Bottom-Up ».

4. Correspondances entre ontologies

4.1. Définition

Il s'agit d'un sous processus commun pour les processus d'alignement et de fusion d'ontologies (en plus d'autres processus supportant l'interopérabilité entre deux ou

plusieurs ontologies). Son objectif est d'identifier les classes similaires entre les ontologies sources. Les classes identifiées comme étant similaires vont être reliées par des correspondances (ou des mappings) lors du processus d'alignement, ou fusionnés en une seule classe lors du processus de fusion d'ontologies. Nous rappelons qu'une ontologie est conçue et développée pour servir comme vocabulaire commun qui est partagé par plusieurs applications et communautés de développeurs de systèmes d'informations, ce qui n'est pas possible avec les BDDs. Le lecteur peut se référer à (Uschold & Gruninger, 2004) pour plus de détails sur les limites des BDDs qui peuvent être contournées par les ontologies. Une ontologie commune est alors enrichie par l'Ingénieur de Connaissances (IC) par des classes et/ou des propriétés spécifiques à l'application et/ou au domaine d'application. Le processus de découverte de correspondances peut être effectivement plus facile s'il est exécuté sur deux extensions qui se réfèrent à la même ontologie commune. En plus, les ontologies sont développées souvent pour faire de l'inférence. Alors, l'inférence et le raisonnement ont un effet important sur le processus de découverte de correspondances entre deux (ou plusieurs) ontologies sources. En se basant sur ces deux aspects, Noy (Noy, 2004) a mis à l'échelle deux architectures principales pour découvrir les mappings entre deux ontologies sources :

- **Utiliser une ontologie partagée,** où l'ontologie commune est définie par les classes et propriétés spécifiques au domaine d'application. Plus ces extensions sont consistantes avec les définitions fournies dans l'ontologie commune, plus facile sera le processus de découverte de correspondances entre les ontologies sources. Parmi les ontologies communes les plus connues dans la littérature, on trouve :
  - SUMO, Suggested Upper Merged Ontology (Niles & Pease, 2001), qui est conçue avec l'objectif de développement d'une ontologie commune et standard qui peut être utilisée dans l'interopérabilité entre les sources de données, la recherche d'informations, l'inférence automatique et le traitement du langage naturel.
  - DOLCE (Gangemi et al., 2003) qui est une ontologie globale conçue avec l'objectif de fournir un cadre de travail commun comme étant une référence aux ontologies du web sémantique qui facilite le partage d'informations entre elles. DOLCE a pour but de capturer les catégories d'ontologies visant le langage naturel et la cognition.

➢ **Utiliser les heuristiques et l'apprentissage automatique,** les approches de découverte de mappings ou de correspondances basées sur les heuristiques pour le processus d'alignement d'ontologies sont similaires aux approches de découverte de matchings pour les schémas de BDDs ainsi qu'aux processus de découverte de matchings des structures XML, où les définitions des composantes lexicales et structurelles sont utilisées (Madhavan, Bernstein, & Rahm, 2001; Rahm & Bernstein, 2001). Les approches basées sur les ontologies sont souvent plus sophistiquées parce qu'elles utilisent la sémantique contenue dans les ontologies, (sémantique des relations is-a, part-of, attachement de propriétés aux classes, les définitions des domaines et co-domaines de propriétés, etc.). Contrairement aux schémas des BDDs, les ontologies possèdent beaucoup plus de contraintes spécifiques. Ces dernières fournissent des bases importantes pour les méthodes de découverte de mappings automatiques.

4.2. Techniques de découverte de correspondances

La plupart des opérations effectuées sur les ontologies, tel que l'Alignement, la Fusion, la Translation, la Coordination, etc. sont conçues et développées à la base de la phase de découverte de correspondances. Cette dernière a pour objectif d'identifier les classes similaires entre les ontologies sources. Plusieurs travaux proposant différentes techniques de découverte de correspondances entre des ontologies qui existent dans la littérature. Selon le type de données analysées lors de ces techniques, ces dernières sont classifiées en techniques intentionnelles et techniques extensionnelles :

4.2.1. Les techniques intentionnelles

Elles sont basées sur l'analyse du contenu interne des ontologies. Il (le contenu) se compose de l'ensemble de classes ainsi que de leurs propriétés qui représente l'ensemble des attributs (data type properties) et l'ensemble de relations qui lient une classe avec une ou plusieurs autres classes (object properties). Les techniques intentionnelles sont elles mêmes classifiées en deux types, les techniques terminologiques et les techniques structurelles :

4.2.1.1. Les techniques terminologiques

Ce type de technique se concentre sur l'analyse et la comparaison des termes de type « chaîne de caractères » représentant les noms des classes, de leurs attributs (object properties) ou de leurs relations (data type properties). Ces termes peuvent être

analysés lexicalement ou sémantiquement. Ceci donne alors lieu à deux types d'analyse terminologiques :

i. Les techniques terminologiques lexicales : Le moyen le plus simple pour comparer les noms de deux entités ontologiques (de même type) est de mesurer la similarité entre eux. Un nom d'une entité ontologique est lexicalement, soit simple, composé d'une seule chaîne de caractères, soit composé, comportant plusieurs sous chaînes de caractères. Alors, avant de procéder au calcul de similarité, les techniques lexicales commencent par les normaliser à travers une lemmatisation pour extraire la chaîne de caractères principale qui représente les noms à comparer. Plusieurs techniques terminologiques lexicales existent dans la littérature. A titre illustratif et non exhaustif, nous citons la distance de Jaro, Jaro Winkler, Levinstein, Hamming, l'indice et la distance de Jaccard et la distance TF/IDF :

a) La distance de Jaro (Jaro, 1989)**:** Elle mesure la distance entre deux chaînes de caractères. Elle est principalement utilisée pour détecter les doublons. Plus la distance de Jaro entre deux chaînes de caractères est élevée plus les deux chaînes de caractères sont similaires. Cette mesure est particulièrement adaptée au traitement de chaînes courtes comme des noms ou des mots de passe. Le résultat est normalisé de façon à avoir une valeur entre 0 et 1. Le 0 indique que les deux chaînes de caractères sont totalement différentes et 1 qu'elles sont totalement similaires. La distance de Jaro entre deux chaînes de caractères S_1 et S_2 est définie par :

$$d_{jaro}(s_1, s_2) = \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right)$$

Où $|s_1|$ (resp. $|s_2|$) est la longueur de la chaîne de caractères s_1 (resp. s_2). m : est le nombre de caractères correspondants. t =N/2 : est le nombre de transpositions. N : est le nombre de couples de caractères correspondants qui ne sont pas dans le même ordre dans leurs chaînes respectives.

Deux caractères identiques de s_1 et s_2 sont considérés comme correspondants si leur éloignement (la différence entre leurs positions dans leurs chaînes respectives) ne dépasse pas la valeur *val* : $val = \frac{\max(|s1|,|s2|)}{2} - 1$

b) La distance de Jaro-Winkler (Winkler, 2006)**:** Elle est dérivée de la distance de Jaro. Elle utilise un coefficient de préfixe *p* et favorise les chaînes commençant par un préfixe de longueur *l* (avec $l \leq 4$). La distance de Jaro Winkler entre deux chaînes de caractères s_1 et s_2 est définie par :

$$d_{jaro-winkler} = d_{jaro} + \left(l * p\left(1 - d_{jaro}\right)\right)$$

où *l* : est la longueur du préfixe commun (max 4 caractères). *P* : le coefficient qui permet de favoriser les chaînes avec un préfixe commun. Winkler propose une valeur de $p=0.1$.

c) La distance de Levinstein (Levenshtcin, 1966)**:** La distance de levinstein entre deux chaînes de caractères x et y est définie comme étant le nombre minimal d'opérations d'édition pour transformer x en y par Substitution d'une lettre de x par une lettre de y et/ou Suppression d'une lettre de x et insertion d'une lettre de y. En d'autres termes, elle est égale au nombre minimal de caractères qu'il faut supprimer, insérer ou remplacer pour passer d'une chaîne à l'autre. Elle est alors, d'autant plus grande que le nombre de différences entre les deux chaînes est grand.

d) La distance de Hamming (Hamming, 1950)**:** La distance de Hamming fournit un moyen simple mais pas toujours pertinent pour comparer deux chaînes de caractères. Elle mesure la distance entre deux chaînes de caractères x et y de même longueur, comme étant le nombre de positions dans lesquelles les deux chaînes de caractères possèdent des lettres différents:

$$d_{ham\,ing}(x,y) = card\left\{ i / x[i] \neq y[i] et 0 \leq i \leq -1 \right\}$$

e) La distance d'Edition (Klein & Fensel, 2001): Pour transformer une chaîne de caractère x en une autre y, on peut utiliser trois opérations de base, la suppression, l'insertion et la substitution. La distance d'Edition entre deux chaînes de caractères est définie comme étant le nombre minimal d'opération d'Edition requises pour transformer l'une vers l'autre. De même, les coûts de chacune de ces opérations ont été définis. La distance d'Edition sert alors à trouver les opérations de conversion d'une chaîne de caractères vers une autre avec le coût minimal. En prenant les coûts de suppression et d'insertion égaux à 1 et le coût de substitution égale à 2, la distance d'Edition est définie par :

$$1 - DE(ax, by) = DE(a, b), si \rightarrow x = y$$
$$2 - DE(ax, by) = \min(DE(a, b) + 2, DE(ax, b) + 1, DE(a, by) + 1), si \rightarrow x \neq y$$
$$3 - DE(a, ¤) = |a| etDE(¤, b) = |b|$$

Quand tous les coûts sont égaux à 1, nous tombons sur la distance de Levinstein. La distance d'Edition peut être considérée comme étant une généralisation de la distance de Hamming.

f) L'indice et la distance de Jaccard (Hunter, Drennan, & Little, 2004) : Il s'agit de deux métriques utilisées en statistique pour comparer la similarité et la diversité entre les échantillons. L'indice ou encore le coefficient de Jaccard est le rapport entre la cardinalité (la taille) de l'intersection des ensembles comparés et la cardinalité de l'union des ensembles. Il permet d'évaluer la similarité entre les ensembles. Soit deux ensembles A et B, l'indice de Jaccard est défini par :

$$Indice_{Jaccard} = \frac{|A \cap B|}{|A \cup B|}$$

L'indice de Jaccard mesure la dissimilarité entre les ensembles. Elle consiste tout simplement à soustraire l'indice de Jaccard de 1 :

$$D_{Jaccard} = 1 - Indice_{Jaccard} = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}$$

g) La distance TF/IDF (Karen Sparek, 2004)**:** De l'anglais : Term Frequency/ Inverse Document Frequency. Est une méthode de pondération souvent utilisée en recherche d'informations. Elle permet d'évaluer la pertinence d'un mot dans un document d'un corpus. Elle est alors utilisée au cas où on a besoin de comparer des descriptions textuelles tel que les commentaires. Dans ce cas les portions de texte sont décomposées en unités appelées « Token ». Le poids augmente proportionnellement avec le nombre d'occurrence de mots dans le document. La fréquence d'un terme est simplement le nombre d'occurrences de ce terme dans le document. Alors que la fréquence inverse de document est une mesure de l'importance du terme dans l'ensemble du corpus. Dans le schéma TF/IDF, elle vise à donner un poids plus important aux termes les moins fréquents, considérés comme plus discriminants. Les mesures TF/IDF sont définies comme suit :

$$TF = \frac{n(t)}{N}$$

$$IDF = \log\left(\frac{|D|}{d(t)}\right)$$

tel que : *D* : Un corpus de documents. |*D*| : Le nombre de documents dans le corpus D. *n(t)* : Le nombre d'occurrences du terme t dans le document. *N* : Le nombre total de termes dans le document.

d(t) : Le nombre de documents qui contiennent au moins une fois le terme *t*.

ii. Les techniques terminologiques sémantiques : Les mesures proposées jusqu'ici ne garantissent la satisfaction d'aucune contrainte sémantique. En effet, deux (ou plusieurs) chaînes de caractères peuvent être, lexicalement, totalement différentes mais avoir le même sens (synonymes). Pour contourner ce problème de contradiction, plusieurs techniques terminologiques basées sur la sémantique ont été proposées dans la littérature. Ces techniques sont basées sur l'interprétation des termes comparés dans la langue analysée et/ou sur des BDDs lexicales générales ou spécifiques à un domaine particulier. Parmi les BDDs lexicales générales (ontologies générales) les plus utilisées nous citons wordNet. Elle recense l'ensemble de tous les liens sémantiques qui peuvent être entretenus par deux termes dans une langue. Parmi ces liens sémantiques nous citons :

a) L'hyponymie **:** Mot dont le sens est plus spécifique que celui d'un autre. Par exemple, « cheval est hyponyme d'un animal ».

b) L'hyperonymie **:** Mot dont le sens inclut celui d'un autre. Par exemple, «fruit est hyponyme de pomme ».

c) L'holonomie **:** Terme lié à un autre terme par une relation « tout à partie ». Par exemple, «corps est holonyme de bras ».

d) Le meronymie **:** Opposé de l'holonomie, terme lié à un autre terme par une relation « partie à tout ». Par exemple, «bras est meronyme de corps ».

e) La synonymie **:** Mot dont le sens est le même que celui d'un autre mot. Par exemple, «automobile est le synonyme de voiture ».

f) L'antonymie **:** Mot dont le sens est opposé à celui d'un autre mot. Par exemple, «petit est l'antonyme de grand ».

g) La paronymie **:** Mot dont la ressemblance phonétique avec un autre mot entraîne plusieurs confusions. Par exemple, «irruption est paronyme avec éruption ».

h) L'homonymie **:** Un seul mot lexical avec différent sens. Par exemple, «tu sais qu'il s'est tu ? (tu, tu) ».

iii. **Les techniques structurelles:** Les techniques structurelles sont basées sur l'analyse des relations sémantiques entre les entités d'un même type dans une ontologie. (Shvaiko & Euzenat, 2005) fait la distinction entre deux types de méthodes structurelles selon le type d'entités analysées :

a) Les techniques structurelles internes : Ces techniques comparent les structures internes des entités telles que les valeurs de cardinalités et le co-domaine de leurs attributs.

b) Les techniques structurelles externes : Ces techniques comparent les relations entre les entités des ontologies (hyponymie, hyperonymie, …). Elles exploitent un ensemble d'heuristiques tel que si deux entités sont similaires alors leurs voisins sont également similaires d'une certaine façon.

4.2.1.2 Les techniques extensionnelles

Ces techniques sont exploitables avec les ontologies peuplées (populated). Une ontologie peuplée est une ontologie équipée d'un ensemble d'instances de ses classes. Les techniques extensionnelles mesurent alors la similarité entre deux classes de deux ontologies différentes à travers le calcul de similarité entre les deux ensembles de leurs instances. L'apprentissage automatique est souvent exploité par les techniques extensionnelles.

4.3 Stratégies de combinaison de techniques

La plupart des systèmes de découverte de correspondances combinent différentes techniques pour renforcer leurs performances. Les techniques peuvent être combinées par l'agrégation des résultats de distances (Van Hage, Katrenko, & Schreiber, 2005) en utilisant des fonctions de sélection pour choisir laquelle de ces distances est à utiliser dans le cas présent (Jian et al., 2005; Tang et al., 2006) ou par l'intégration de toutes les mesures lors du calcul d'une distance globale (Euzenat & Valtchev, 2004; Melnik, Garcia-Molina, & Rahm, 2002). Il est à noter qu'il y a une différence quand les ensembles d'apprentissage (les instances de classes) sont disponibles ou pas. Ceci est

généralement utilisé quand un algorithme de découverte de correspondances est exigé pour reconnaître les instances. Si les instances de classes sont disponibles, les techniques d'apprentissage automatiques sont applicables tel que l'algorithme d'apprentissage de Bayes, les SVM (Support Vector Machine) ou encore les arbres de décision.

Parfois le contenu informationnel des ontologies n'est pas suffisamment large pour garantir des prédictions de similarité performantes. Les bases de données lexicales (recensant les liaisons sémantiques entre les entités ontologiques) spécialisées dans un domaine particulier peuvent ne pas être exploitables pour d'autres domaines. En plus, les BDDs générales peuvent ne pas recenser tous les liens sémantiques entre les termes spécifiques au sein d'un domaine particulier. Ces problèmes et d'autres, peuvent abaisser les performances de prédiction de similarité de techniques intentionnelles. D'un autre côté, si les ontologies comparées ne sont pas des ontologies peuplées (équipées de leurs instances), ces techniques (intentionnelles) ne sont pas du tout applicables. De plus, il n'existe pas toujours des objets qui sont, en même temps, des instances pour des classes similaires. Ce qui réduit considérablement les performances de prédiction de similarités des techniques extensionnelles. Alors, pour contourner ces problèmes et d'autres, la combinaison inter et intra techniques intentionnelles et structurelles est une solution prometteuse. A ce stade, trois formes de combinaison de stratégies sont mises au point : la combinaison séquentielle (ou linéaire), la combinaison parallèle (on non-linéaire) et la combinaison hybride.

4.3.1 La combinaison séquentielle ou linéaire

Il s'agit de la composition la plus simple pour combiner différentes techniques. Un ordre spécifique de l'application des différentes techniques doit être préfixé. Les prédictions de dissimilarité sont raffinées successivement jusqu'à la dernière technique, alors que les prédictions de similarité sont rassemblées successivement jusqu'à la dernière technique, également. La combinaison séquentielle ou linéaire est illustrée par la figure 2.4 :

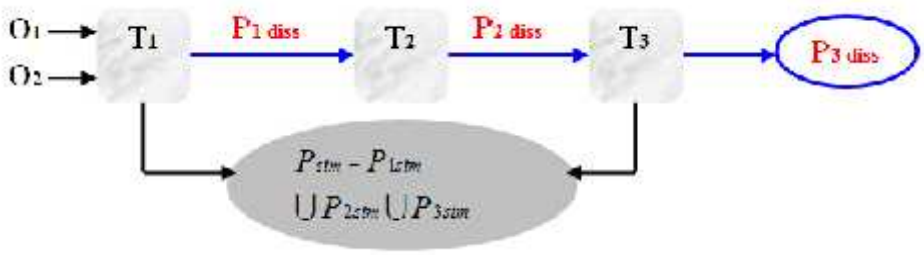


Figure 2-4. La combinaison séquentielle ou linéaire.

4.3.2 La combinaison parallèle ou non linéaire

Il s'agit d'exécuter parallèlement et séparément toutes les techniques. Ensuite, on se base sur une certaine fonction d'agrégation des résultats de toutes les techniques pour aboutir à une prédiction finale. La fonction d'agrégation peut considérer les résultats de la technique produisant les résultats les plus élevés, comme elle peut considérer la moyenne pondérée des résultats de toutes les techniques en affectant des poids différents selon la pertinence de la technique. La combinaison parallèle est illustrée par la figure 2.5:

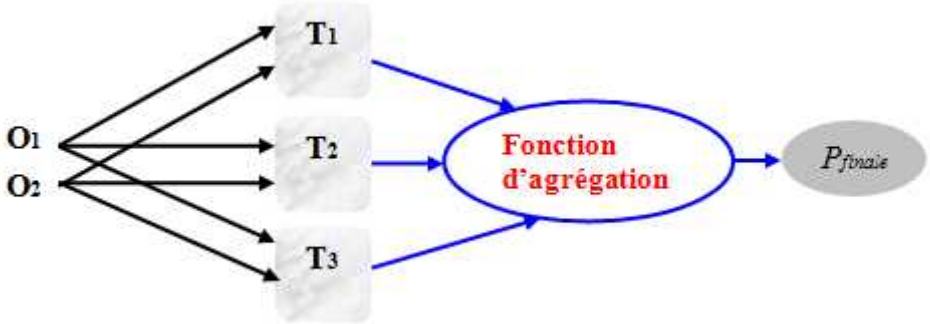


Figure 2-5. La combinaison parallèle ou non linéaire.

4.3.3 La combinaison hybride: Il s'agit de combiner plusieurs compositions séquentielles et parallèles à la fois comme l'exemple du modèle illustré par la figure 2.6:

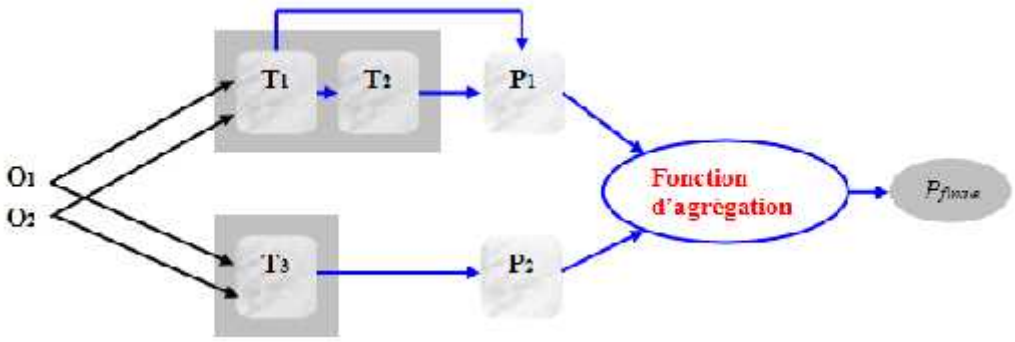


Figure 2-6. La combinaison hybride.

4.4 Exploitation des correspondances

Une fois les correspondances entre ontologies sources obtenues, elles peuvent être exploitées pour accomplir différentes techniques tel que mentionné par (Euzenat, Mocan, & Scharffe, 2008) , à savoir : la réécriture de requête, la transformation d'instances et la fusion d'ontologies.

4.4.1 Réécriture de requêtes (Query rewriting)

Une requête adressée à une ontologie source o_1 a besoin d'être réécrite en termes d'une requête pour une ontologie cible o_2. En d'autres termes, le moteur de réécriture de requêtes reçoit en entrée la requête d'origine « q », les correspondances entre o_1 et o_2 et retourne une requête « q_2 » en termes de o_2.

4.4.2 La transformation d'instances

Un ensemble d'instances décrites sous une ontologie source o_1 a besoin d'être transformé en un ensemble d'instances décrites en termes d'une ontologie cible o_2. Cette transformation est accomplie en utilisant les correspondances ente les deux ontologies en entrée. Les nouvelles instances de classes de o_2 sont décrites et les valeurs de leurs attributs sont transformées selon les correspondances (Scharffe & De Bruijn, 2005). Ce processus peut créer plusieurs instances cibles pour une seule instance source et inversement, pour plusieurs instances sources il peut créer une seule instance cible.

4.4.3 La fusion d'ontologies

Un ensemble d'ontologies sources ont besoin d'être fusionnées en une seule ontologie. Le processus de fusion d'ontologies peut être complètement automatique, si l'ensemble des correspondances adéquates est découvert (Scharffe, 2007).

5 Évaluation d'un processus de fusion d'ontologies

5.1 Critères d'évaluation

Le processus d'évaluation des outils de fusion d'ontologies est un processus qui se base sur la comparaison de plusieurs outils de fusion afin de trouver l'outil le plus performant. Mais, les outils de fusion d'ontologies se différencient les uns des autres selon plusieurs critères : le type des entrées, le type des sorties, le langage de représentation des ontologies, le rôle de l'utilisateur lors de la fusion, etc. Il est donc, impossible de comparer ces outils sans en tenir compte. Souvent, la décision qu'un outil est meilleur qu'un autre dépend principalement de l'utilisateur. C'est lui qui décide quel outil est meilleur pour son travail, ou si un outil particulier est plus adapté pour son travail en fonction du langage de représentation des ontologies sources, des résultats qu'il attend de la fusion, de l'effet de la précision des résultats et de l'interface par laquelle il peut interagir avec l'outil lors de la fusion. Dans (Noy & Musen, 2002), les auteurs ont mis au point un certain nombre de critères sur lesquels, un utilisateur doit se baser pour choisir l'outil qui lui correspond le mieux.

- **Les exigences d'entrées**

L'outil de fusion, exige-t-il un paradigme de représentation de connaissances particulier ? Quels sont les éléments des ontologies sources que le système utilise ? Parmi ces éléments, lesquels sont exigés par l'outil de la fusion ? Cette information peut contenir les noms de classes, la hiérarchie de classes, les définitions des slots, les valeurs de facets, les valeurs de slots ou les instances ?

- **Niveau d'interaction avec l'utilisateur**

L'outil de fusion, fait-il les comparaisons en mode « Batch », en d'autres termes, les résultats de comparaison sont-ils présentés en fin d'exécution ou l'outil est-il interactif où les résultats intermédiaires sont analysés par l'utilisateur et l'outil se base sur un retour en arrière pour les analyses futures ?

- **Type de sortie**

Les résultats de l'outil de fusion sont-ils de quel type ? Un ensemble de règles d'articulation ? Une ontologie fusionnée ? Deux ontologies reliées par un bridge sémantique représentant les mappings entre elles ? Une liste de couples de classes reliés (avec un certain degré de confidence) ?

- **Contenu des sorties**

Quels éléments des ontologies sources sont reliés dans l'ontologie de la sortie ? Ces éléments peuvent contenir des relations entre classes, slots, valeurs, ou instances.

Alors, comme déjà suggéré, il n'y a pas un outil qui soit meilleur qu'un autre. Mais, selon les besoins, un outil est plus approprié pour un utilisateur particulier qu'un autre. Si les ontologies sources de l'utilisateur contiennent des instances, les outils qui analysent ce type de données lors de leur exécution sont plus performants. En revanche, si les ontologies sources ne contiennent pas d'instances, ce type d'outil est inapproprié. De même, si les besoins de l'utilisateur sont limités à une approximation des mappings entre ontologies sources, les outils fournissant des performances moins élevées mais qui ne demandent que peu d'intervention humaine soient plus appropriés, etc.

5.2 Processus de fusion Vs processus de construction d'ontologie

La fusion d'ontologies est le processus de développement d'une nouvelle **ontologie à partir de deux (ou plusieurs) ontologies sources** en fusionnant les entités similaires, et en ajoutant directement les entités différentes de façon à préserver l'ordre hiérarchique des classes sources dans l'ontologie résultat de la fusion. Alors que la construction d'ontologies est le processus de développement d'une nouvelle ontologie **à partir du zéro**. D'un autre côté, **les outils de développement d'ontologies** sont des environnements d'édition d'ontologies, capable d'éditer (ajouter, supprimer ou modifier) tous les types d'entités ontologiques, en l'occurrence, les classes, les propriétés (propriétés d'objet ou propriétés de type de données) et les instances. Ils ont le potentiel d'importer, étendre et exporter des ontologies. Les éditeurs d'ontologies sont souvent réalisés par des navigateurs permettant de visualiser graphiquement les hiérarchies d'ontologies, en plus des moteurs de recherche internes permettant de chercher les entités ontologiques (classes, propriétés et instances) vérifiant certains critères ou contraintes. Les outils de développement d'ontologies les plus connus dans la littérature sont cités dans le chapitre précédent. Les outils de fusion d'ontologies existant dans la littérature sont tous fondés sur les résultats de leurs sous processus de découverte de similarité et de différences entre ontologies sources. Ils sont soit complètement automatiques ne demandant aucune intervention humaine, soit semi-

automatiques fournissant un environnement qui assiste l'utilisateur à identifier, définir et valider les correspondances entre ontologies sources et/ou à construire la hiérarchie de l'ontologie résultat de la fusion. Les outils de fusion d'ontologies les plus connus dans la littérature sont cités dans la section 7 de ce chapitre. Il est à noter que certains outils de fusion d'ontologies sont des extensions d'autres outils de construction d'ontologies, tel que Prompt, Prompt-Diff, Anchor Prompt, etc. D'un point de vue critères d'évaluation, ils existent plusieurs cadres de travail pour évaluer les outils de développement d'ontologies. Dans (Duineveld et al., 2000), les auteurs ont utilisé différents outils de développement pour construire la même ontologie de domaine à fin de comparer leurs performances. Dans (Noy & Musen, 2002), les auteurs ont extrait quelques critères d'évaluation et aspects de comparaison à partir des cadres de travail existant dans la littérature, à savoir :

- L'interopérabilité avec d'autres outils de développement d'ontologies en plus de la possibilité d'importer et d'exporter les ontologies dans différents langages de représentation.
- L'expressivité des modèles de connaissances.
- L'évolutivité (scalability) et l'extensibilité.
- La disponibilité des services d'inférences et la capacité d'insérer de nouvelles connaissances.
- L'utilisabilité des outils de développement.

D'un autre côté, et vu que le processus de fusion d'ontologies est aussi un processus de développement d'une nouvelle ontologie (mais pas à partir de zéro), plusieurs critères d'évaluation des outils de développement des ontologies sont réutilisés pour évaluer les outils de fusion d'ontologies. A titre illustratif et pas exhaustif, l'expressivité, l'évolutivité (scalability) et l'extensibilité des langages de représentation sont des facteurs très intéressants. Mais, en détaillant, on remarque que l'évaluation d'un processus de fusion d'ontologies diffère beaucoup de celle d'un processus de développement d'ontologies. En effet, tous les outils de développement d'ontologies s'exécutent sur des entrées très similaires (un domaine d'application, et optionnellement un ensemble d'ontologies à réutiliser et un ensemble de critères à satisfaire) et l'outil de construction d'ontologies produit toujours en sortie, une

ontologie vérifiant les critères préexistants. A l'opposé de ces outils, les outils de fusion d'ontologies peuvent avoir différents types d'entrées en différents langages de représentation et produire des sorties de différents formats de représentations, en plus de plusieurs autres critères. (Voir les sections 7.1, 7.2 et 7.3, comparaison des outils de fusion).

6. Applications de la fusion d'ontologies

Dans ce qui suit, nous présentons quelques applications de fusion d'ontologies existant dans la littérature :

6.1. Fusion d'ontologies approximatives pour le web sémantique

Dans cette application, les auteurs (Richardson & Mazlack, 2004) ont mis au point un processus composé de trois étapes majeures : D'abord, construire (ou découvrir) les ontologies initiales en utilisant les techniques des langages naturels pour examiner les pages web individuelles. Ensuite, l'ontologie obtenue, est agrandie pour qu'elle puisse contenir les nouvelles informations. Enfin, fusionner les ontologies ainsi construites pour obtenir de nouvelles ontologies qui présentent précisément, les structures représentées par les ontologies individuelles. L'ontologie fusionnée est alors plus large et elle offre ainsi des opportunités plus grandes pour accéder aux données.

6.2. Fusion des ontologies de maintenance de logiciels (Software)

En s'inspirant de la méthode PROMPT, les auteurs (Vizcaíno et al., 2005) proposent de fusionner deux ontologies complémentaires du même domaine de maintenance, dont une est développée par Dias et al. en 2003. Elle identifie les connaissances nécessaires lors d'un processus de maintenance et est composée de cinq sous ontologies. La deuxième ontologie est proposée par Ruiz et al. en 2004 occupée de la gestion des projets de maintenance de logiciel et est composée de quatre sous ontologies. Vu que la décomposition en sous ontologies présente quelques intersections, les auteurs décident de travailler itérativement, fusionner les sous ontologies au lieu de fusionner les ontologies complètes.

6.3. Fusion d'ontologies dans une optique PLM

Dans cette application, l'auteur (Mostefai, 2008) a proposé de fusionner les deux ontologies TOVE et ENTREPRISE qui sont des ontologies décrivant l'entreprise industrielle. L'auteur s'est intéressée à fusionner certaines parties de ces ontologies

relatives à la description de produits représentant des pièces mécaniques afin de produire une représentation PLM (Product Lifecycle Management, pour la Gestion de Cycle de vie d'un Produit) des produits concernés. Dans cette application, les deux ontologies sont alors fusionnées en utilisant l'algorithme de fusion PROMPT.

6.4. Fusion d'ontologies linguistiques, globale et spécialisée

Dans cette application, les auteurs (Magnini & Speranza, 2002) **o**nt entamé le problème de l'interopérabilité des ontologies linguistiques spécialisées à travers leur intégration dans des ontologies linguistiques globales. Selon les auteurs, la possibilité de fusionner des informations à différents niveaux de spécificité est une exigence essentielle, en particulier dans les domaines économique et judiciaire qui partagent une quantité significative de termes communs qui peuvent être inclus dans une ontologie commune. L'approche ainsi proposée est réalisée à travers une procédure semi-automatique ayant 4 étapes majeures : l'Identification de correspondances basiques, l'alignement, la fusion et la résolution des inconsistances**.**

7. Travaux connexes

Plusieurs algorithmes et outils d'alignement et/ou de fusion d'ontologies existent dans la littérature. Dans ce qui suit, nous survolons la littérature des travaux connexes tout en décrivant trois catégories d'algorithmes d'alignement et/ou de fusion d'ontologies. La première catégorie renferme les algorithmes semi-automatiques, la deuxième décrit les algorithmes complètement automatiques et la troisième représente les algorithmes dont les logiciels (software) ainsi que les résultats analytiques sont téléchargeables à partir du site web de la campagne OAEI'2015. Ces trois catégories vont nous servir de références pour positionner notre contribution dans les chapitres quatre et cinq.

Première catégorie : Algoritmes semi-automatiques

7.1.1. PROMPT Suite

PROMPT Suite (Noy & Musen, 2003) constitue un cadre de travail pour gérer les ontologies. Il est composé de différents outils de gestion d'ontologies qui sont des plugins sous Protégé 2000. Il s'agit de iPROMPT (Interactive PROMPT, un outil de fusion d'ontologies), Anchor Prompt (un outil d'identification des classes reliées servant comme des suggestions pour d'autres outils tel que iPROMPT), PROMPT Diff

(un outil de gestion de différentes versions d'une ontologie), PROMPT Factor (un outil de factorisation sémantique des sous ontologies).

iPROMPT, Initialement connu par **Smart** (Noy & Musen, 1999), et ensuite par **PROMPT** (Noy & Musen, 2000) est un outil interactif de fusion d'ontologies sous forme d'un plugin sous l'environnement *Protégé 2000. Il s'agit* d'un outil interactif orienté utilisateur, qui aide à fusionner des ontologies hétérogènes. PROMPT propose à l'utilisateur des suggestions de fusion (*ToDo List*). Puis, l'utilisateur lance une des opérations (alignement, fusion, comparaison, etc.) soit à partir de la liste des propositions suggérées par PROMPT, ou en utilisant l'environnement d'édition d'ontologie pour spécifier directement l'opération voulue. Ensuite, PROMPT exécute automatiquement l'opération lancée par l'utilisateur, tout en adaptant les changements additionnels basés sur le type de l'opération, détermine les conflits introduits par cette opération (*Conflict List*), tout en proposant des solutions, et enfin génère une liste des suggestions à l'utilisateur en fonction de la nouvelle situation et de la structure courante de l'ontologie.

7.1.2. Chimaera

Chimaera (McGuinness et al., 2000) est un outil de fusion et de diagnostic d'ontologies, développé au sein du laboratoire KSL à l'université de Standford. L'utilisateur interagit avec Chimaera à travers un navigateur web tel que NetScape ou internet explorer, via une interface utilisateur basée web. A l'instar de PROMPT, Chimaera est aussi un outil interactif et un plugin dans un environnement de développement, Ontolingua. Mais, les deux diffèrent dans le type de propositions qu'ils suggèrent aux utilisateurs, pour prendre des décisions affectant la fusion. Les deux tâches principales offertes par chimaera sont:

1. Fusionner deux termes sémantiquement identiques à partir de différentes ontologies de façon à être référencés par le même nom dans l'ontologie résultat.
2. Identifier les termes qui doivent être reliés par des relations de subsomption, de disjonction ou par des relations entre les instances tout en offrant un support pour introduire ces relations. Chimaera dispose aussi d'un support pour vérifier, valider et critiquer des ontologies. En se basant sur les similarités de noms (des classes ou des slots), Chimaera génère des listes de résolutions de noms à travers lesquelles, il

propose des termes candidats à être fusionnés ou hiérarchisés qui n'ont pas encore été considérées dans l'ontologie résultat de la fusion. Sur la base de ces listes, l'utilisateur décide ce qu'il doit faire. Chimaera génère aussi une liste de résolution de hiérarchie, à travers laquelle, il propose des portions de la hiérarchie qui sont candidats pour une réorganisation. Chimaera fournit un support pour fusionner les slots, et supporte la fusion de facets, relations, fonctions, instances et axiomes.

7.1.3. ONION (ONtology compositION).

Deux types d'ontologies peuvent être distingués dans le système ONION (Mitra, Wiederhold, & Kersten, 2000; Mitra & Wiederhold, 2001), les ontologies individuelles (ou encore ontologies sources) et les otologies d'articulation contenant les classes et les relations exprimées sous forme de règles d'articulation, (des règles qui expriment les liens entre les domaines). Les règles d'articulation indiquent quelles sont les classes qui sont individuellement ou en conjonction, reliées dans les ontologies sources. L'alignement entre ontologies est exécuté par l'algèbre d'ontologies (Mitra & Wiederhold, 2001; Wiederhold, 1994b). Cette algèbre est constituée de trois opérations, l'intersection, l'union et la différence. L'opération d'intersection produit un graphe d'ontologie qui est l'intersection de deux ontologies sources. L'intersection détermine une portion de la base de connaissances reflétant les classes similaires. L'opération d'Union génère un ensemble unifié contenant les deux graphes d'ontologies originales connectés par des articulations. L'union présente une ontologie unifiée de contenu sémantique cohérent, bien sûre si les ontologies sources sont consistantes. L'opération de différence (o_1-o_2) décrit la différence entre les deux ontologies et est définie comme étant un ensemble de classes et de relations d'o_1 qui n'existent pas dans o_2. L'architecture d'ONION est constituée de quatre composants : Les couches de données, un viewer, un système de requêtes et un moteur d'articulation. La couche de données contient les interfaces des sources externes ainsi que les ontologies d'articulation qui constituent le pont sémantique (semantic bridge) entre les sources. Le viewer est l'interface utilisateur qui visualise et la source et les ontologies d'articulation. Le système de requêtes interprète les requêtes formulées sous forme d'ontologies d'articulation vers un plan d'exécution de requêtes et exécute la requête.

Le moteur d'articulation utilise les règles d'articulation proposées et génère les ensembles des règles d'articulation qui sont retournées vers l'expert pour confirmation.

7.1.4. Glue

Glue (Doan et al., 2004) utilise les techniques d'apprentissage automatique pour créer, semi-automatiquement, des mappings de type (1-1) entre ontologies hétérogènes représentées sous formes de hiérarchies de classes. La similarité de deux classes A et B appartenant à deux hiérarchies O_1 et O_2 respectivement est basée sur l'ensemble des instances en commun des deux classes. L'architecture générale du système Glue est constituée de trois phases :

L'estimateur de distribution : Reçoit en entrée les deux taxonomies O_1 et O_2 ensemble avec leurs instances et applique la technique d'apprentissage automatique pour décider, pour chaque instance de B, si elle est aussi une instance de A ou non. Il est possible d'intégrer plusieurs classifieurs en utilisant un méta classifieur qui utilise une fonction d'agrégation de toutes les prédictions en une seule prédiction totale.

L'estimateur de similarité : Applique une fonction fournie par l'utilisateur tel que le coefficient de Jaccard ou le pps (le parent le plus spécifique) et calcule une valeur de similarité entre chaque pair de classes ($A \in O_1$, $B \in O_2$).

Labeler de relaxation : Reçoit en entrée les valeurs de similarité entre les classes de taxonomies O_1 et O_2 et cherche la configuration du meilleur mapping en utilisant des contraintes spécifiques au domaine fournie par l'utilisateur, conjointement avec un ensemble d'heuristiques.

7.1.5. FCA-Merge

FCA-Merge (Stumme & Maedche, 2001) est un algorithme de fusion d'ontologies basée sur la technique Bottom-Up. FCA-Merge reçoit en entrée, les deux ontologies sources O_1 et O_2 et un ensemble de documents décrivant le domaine d'application des deux ontologies O_1 et O_2 et nécessitant l'ontologie fusionnée O_3. Les instances sont automatiquement extraites à partir des documents en entrée. Par cette étape d'acquisition automatique de connaissances, le contexte formel qui indique quelle classe ontologique apparaît dans quel document, est calculé pour chaque ontologie. La deuxième étape consiste à exécuter l'algorithme de la fusion qui fusionne les deux contextes formels et calcule le treillis de classes à partir du contexte fusionné en

utilisant les techniques d'analyse de classes formelles. L'extraction d'instance ainsi que l'algorithme FCA-Merge sont complètement automatiques. La dernière étape qui consiste à dériver l'ontologie fusionnée à partir des treillis de classes demande une interaction avec l'expert humain. Basé sur le treillis de classes élagué ainsi que l'ensemble des noms de relations R_1 et R_2, l'ontologiste génère les classes et les relations de l'ontologie fusionnée. En fin, les outils graphiques de l'environnement OntoEdit sont utilisés pour supporter le processus FCA-Merge pour générer le format graphique de l'ontologie finale.

7.1.6. OntoMerge

OntoMerge (Dou, McDermott, & Qi, 2005) est une approche de translation d'ontologies en utilisant la fusion d'ontologies et le raisonnement automatique. Pour la translation d'un ensemble de données : OntoMerge accepte un ensemble de données sous forme d'un fichier DAML comme étant une ontologie source O_1. Il va ensuite répondre par un ensemble de données sous forme d'un fichier DAML également mais comme étant une autre ontologie destination O_2. Puis OntoMerge appelle PDDAML qui va translater l'ontologie destination de DAML vers Web-PDDL. Là, un moteur d'ontologie en ligne, OntoEngine télécharge l'ontologie résultat de la fusion des ontologies source et destination (qui doit obligatoirement être manuellement fusionnée et fournie par l'expert humain), en plus de l'ensemble de données à translater. Pour ce faire, il exécute des algorithmes de chaînage avant et arrière et traite tous les effets un par un jusqu'à ce que toutes les inférences soient déduites. Ceci, a pour résultat une nouvelle translation sous forme d'un nouveau fichier en web-PDDL. Un autre appel à PDDAML peut alors translater le fichier résultat en webPDDL vers un fichier DAML.
Dans ce contexte, les ontologies sont fusionnées en considérant leur union où tous les termes sont séparés par leurs espaces de noms. Les axiomes de bridge sont alors utilisés pour relier les parties communes entre les deux ontologies. Lors de la fusion d'ontologies, soit on crée un nouvel espace de noms de l'ontologie fusionnée soit elle utilise l'espace des noms de l'une des deux ontologies (celle qui importe l'autre quand on importe une ontologie dans l'autre).

7.1.7. HconeMerge

HconeMerge (Kotis, Vouros, & Stergiou, 2006) est un outil de fusion des ontologies hétérogènes en se basant sur le calcul d'un morphisme sémantique entre eux. Pour ce faire, HconeMerge suit les étapes suivantes :

1. **Calcul des s-morphismes :** Comme précédemment mentionné, la première étape de Hcone-Merge est le mapping des ontologies sources O_1 et O_2 avec l'ontologie intermédiaire implicite au moyen de S-morphism : $f_i : O_1 \rightarrow O_3$, $f_2 : O_2 \rightarrow O_3$. HconeMerge utilise alors l'approche LSI pour calculer S-morphism. La matrice $(n \times m)$ associée contient les n termes les plus fréquents des m sens de wordNet, ce qui constitue le centre d'intérêt de l'algorithme.

2. **Translation:** En ayant tous les mappings préférés des classes avec les sens de wordNet, les sens prétendus des classes d'ontologies sont capturés. En utilisant les sens informels prétendus des classes, Hcone-Merge interprète les définitions formelles des classes dans un vocabulaire commun et fusionne les définitions interprétées en utilisant les services de raisonnement de la logique de description. En utilisant les sens formels prétendus des classes, Hcone Merge construit une ontologie $O_n(S^n, A^n)$, n= 1, 2, où S^n contient le lexique des sens associés aux classes de l'ontologie $O_n(S^n, A^n)$, *et* A^n contient les relations d'équivalence et d'inclusion interprétées entre les classes correspondantes.

3. **Fusion d'ontologies :** Une fois que les ontologies sources sont alignées (à travers le calcul des morphismes sémantiques entre chaque ontologie source et l'ontologie intermédiaire implicite) la construction actuelle de l'ontologie intermédiaire avec l'ensemble minimal d'axiomes des deux ontologies sources produit l'ontologie fusionnée des ontologies sources.

➢ **Comparaison des travaux:** Dans ce qui suit, nous présentons un résumé récapitulatif des principales différences entre les algorithmes de fusion d'ontologies semi-automatiques, décrits ci-dessus, selon un certains nombre de critères comme indiqué par la table 2 :

Table 2.1. Comparaison des travaux de la première catégorie.

Propriétés		Chimaera	ONION	PROMPT	FCA-Merge	Glue	OntoMerge	Hcone-Merge
Opération		Fusion + diagnostique	Composition	Mapping + fusion	Fusion	Mapping	Fusion + raisonnement + translation	Fusion
Cadre de travail		Ontolingua	Indépendant	Protégé 2000	Indépendant	Indépendant	Indépendant	Indépendant
Langage de représentation		Ontologua	Graphes libellés et orientés + règles de Horn	Rdfs + owl	Taxonomies de classes d'ontologies populaires	Taxonomies	DAML : ontologies sources et fusionnée. Web pddl : raisonnement automatique	Logique de description
Langage de mapping		Mesure de similarité linguistique	Mesure de similarité linguistique. Inférence structurelle basée sur les heuristiques. Règles d'articulation	Analyseur basé heuristiques	Mesure de similarité linguistique + algorithme TITANIC pour calculer le treillis élagué de classes	Approche d'apprentissage multi-stratégies 'technique d'apprentissage automatique).	Pas applicable	s-morphismme
Ressources externes		Aucun	wordNet	Aucun	Aucun	Aucun	Aucun	wordNet
Analyse lexicale		Non	Non	Oui	Oui	Oui	*	Oui
Analyse sémantique		Oui	Oui	Oui	Oui	Oui	*	Oui
Niveau d' alignement	Classes	Non	Oui	Oui	Non	Oui	*	Oui
	Propriétés	Non	Non	Non	Non	Oui	*	Non
	Instances	Non	Non	Non	Oui	Oui	*	Non
	Structure	Non	Oui	Oui	Oui	Oui	*	Oui
Type d'interaction avec l'utilisateur humain		Prendre des décisions	L'expert humain	Accepter, rejeter ou	Choisir la fonction de	Définir les mappings pour	Générer manuellement,	Décider et choisir les

	affectant le processus de la fusion	choisit, supprime ou modifie les mappings proposés en utilisant un outil GUI	ajuster les suggestions du système	calcul de similarité	l'ensemble d'apprentissage. Choisir la fonction de calcul de similarité. Fixer les poids du classifieur. Analyser les mappings suggérés par le classifieur.	l'ontologie fusionnée qui combine les l'ontologie source DAML avec sa nouvelle version translatée en ontologie DAML	stratégies de fusion. Guider le processus en cas d'inconsistance. Valider les sens prétendus des termes.
Entrées	2 fichiers en RDF et DAML	Fichier RDF	2 ontologies DAML(OWL)	2 ontologies en DL + ensemble de documents	2 taxonomies de classes + instances	Une ontologie source en DAML	2 ontologies sources
Sorties	Ontologie fusionnée	Ensemble de règles d'articulation entre 2 ontologies	Ontologie fusionnée	Ontologie fusionnée	Ensemble de paires de classes similaires	Ontologie fusionnée et translatée	Ontologie fusionnée
Architecture de mapping	Point à point Top-down	Point à point Top-down	Point à point Top-down	Point à point Bottom-up	Point à point Top-down	Selon l'expert humain	Point à point Bottom-up
Algorithme/ API utilisé	/	SKAT	OKBC	Concept lattice	Coefficient de Jaccard + MSP	Rien pour la fusion + raisonnement à base de chaînage avant et arrière pour la translation	Nee classic DL
Apprentissage automatique	Non	Non	Oui	Non	Oui	Non	Non
Développé par	Université de Stanford,	Université de Stanford,	Stanford medical	Université de	Université de Wisconsin,	DARPA/DAML program, USA ,	AI lab, greece, 2002

	USA, 2005	USA, 2004	Informatics, Université Stanford, USA, 2003	Karlsruhe, Germany, 2001	2005	2006	

* : Oui ou non, ça dépend des experts humains, car la fusion est purement manuelle dans ces cas.

7.2. Deuxième catégorie : Algorithmes complètement automatiques

7.2.1. Algorithme de Robin et Uma.

Les auteurs (Robin & Uma, 2010) utilisent une stratégie hybride qui se compose de quatre sous stratégies : Alignement lexical, alignement sémantique, vérification de similarité et fonctions d'heuristiques. Cet algorithme reçoit deux fichiers .owl, en entrée. En cherchant les correspondances entre ces deux fichiers, le parcours des ontologies commence du haut vers le bas dans une ontologie et du bas vers le haut dans l'autre. Selon les auteurs, ceci permet d'éliminer plusieurs comparaisons non nécessaires. Cette technique est basée sur le principe que si un nœud feuille d'une ontologie n'as pas de similarité avec le nœud racine de l'autre ontologie, alors les deux ontologies sont complètement différentes. D'abord, l'ontologie de fusion (le fichier .owl en sortie) est initialisée par la première ontologie O_1. Ensuite, pour chaque classe C_2 de O_2, une analyse lexicale entre C_2 et C_1 de O_1 est exécutée. Si l'analyse lexicale échoue à détecter un mapping entre les deux classes en cours, l'analyse sémantique va compléter la tâche. Si un mapping est détecté soit par l'analyse lexicale ou l'analyse sémantique, un module de vérification de similarité entre C_1 et C_2 est lancé. Quand C_2 est comparé avec toutes les classes C_1 de O_1, une fonction d'heuristique (qui est chargée de l'écriture dans le fichier de sortie) est appelée. Elle vérifie si un mapping lexical ou sémantique est trouvé. Si c'est le cas, une telle classe est écrite ainsi que ses propriétés dans le fichier résultat, ensemble avec sa classe correspondante. Sinon, et en utilisant la valeur calculée lors du module de vérification de similarité entre les propriétés, les classes les plus reliées sont identifiées. Et la classe avec le plus grand nombre de propriétés sera la sous classe de l'autre. Ensuite, C_2 est initialisée par la classe précédente (feuille suivante ou classe du niveau suivant) et le processus se réitère jusqu'à la dernière classe (racine) de O_2.

Chaque synonyme de C_1 est alors comparé avec C_2 en utilisant une analyse lexicale.

7.2.2. Algorithme de Maiz et al.

Les auteurs **(Maiz, Boussaid, & Bentayeb, 2008; Maiz et al., 2010)** utilisent la technique de classification hiérarchique et des mécanismes d'inférence pour fusionner les ontologies. Cet algorithme est constitué de quatre étapes principales :

- **La première étape** qui est la classification hiérarchique de classes et construction de $SYN_{i,}$ consiste à collecter l'ensemble de toutes les ontologies, et d'y appliquer un algorithme de classification hiérarchique qui va produire des sous ensembles de classes de classes synonymes SYN_i, où chaque classe va contenir l'ensemble des classes qui sont similaires, en utilisant une mesure de similarité entre classes. A chaque classe SYN_i, est affecté un nom global qui représente le nom des classes classifiées.
- **La deuxième étape** qui est la génération des ensembles SUB_i, consiste à construire pour chaque ontologie, l'ensemble des paires de classes (père, fils) où la première composante de la paire subsume la deuxième.
- **La troisième étape** qui est la génération de l'ensemble SUBg (l'union des SUB_i) qui consiste à fusionner les ensembles SUB_i pour générer l'ensemble globale SUB_g.
- **La dernière étape** qui est la résolution de conflits et construction de SUB_g final consiste à utiliser les classes de synonymes trouvées lors de la première étape avec l'ensemble des paires de classes de la deuxième étape pour résoudre les conflits sémantiques entre les classes dans ce dernier. En d'autres termes, pour enlever les redondances dans SUB_g, on remplace le nom de chaque classe des paires de la liste SUB_g par le nom globale de sa classe, et là, les redondances seront claires et on peut les enlever. Cette étape se termine par la transformation de l'ensemble des paires de classes en un arbre qui représente la hiérarchie de l'ontologie fusionnée.

7.2.3. Algorithme de Cho et al.

Les auteurs **(Cho, Kim, & Kim, 2006)** se sont basés sur une approche horizontale par classification de classes qui analyse les mappings entre ontologies à travers les classes similaires intégrés au même niveau. Une approche verticale par mesure de similarité qui crée des règles à partir des mesures de similarité entre classes intégrées dans différents

niveaux (quand ceci est impossible par l'approche horizontale). L'intégration et la translation de classes en utilisant des approches horizontale/verticale veut dire qu'on peut obtenir des informations à partir des mappings entre classes similaires et la mesure de similarité entre classes. Pour ce faire, l'analyse de relations entre classes en utilisant wordNet est effectuée. Quatre types de relations peuvent être déduits à partir de wordNet. Dans ce contexte, la disjonction, la spécialisation, l'intersection et l'équivalence. Pour obtenir les relations entre classes, les auteurs ont utilisé le contenu d'informations qui est mesuré par le nombre de mots qui sont subsumés par la classe en question. La classification de classes pour intégrer des classes spécifiques et classifier les classes similaires. Il s'agit alors d'intégrer plusieurs classes dans le même niveau. Pour ce faire, on a besoin d'une méthode automatique ou d'un expert humain. Mais, wordNet est utilisé dans ce cas pour résoudre ce problème. Lors de l'analyse de similarité entre classes dans différents niveaux, l'approche horizontale est impossible. Pour résoudre ce problème, l'approche verticale est appelée. Alors, la similarité entre classes est mesurée et les classes sont intégrées dans leurs positions appropriées.

7.2.4. Algorithme de Li et al.

L'idée principale de l'approche de **(Li et al., 2009)** est d'utiliser un algorithme de similarité d'ontologies pour déterminer les relations entre les ontologies locales. Ensuite, bénéficier de l'avantage de treillis de classes (concept lattice) pour décrire les relations logiques entre les ontologies locales et éventuellement, construire l'ontologie globale et donner les relations entre ontologie globale et les ontologies locales. Soit deux systèmes A et B, chacun est composé de plusieurs ontologies locales et hétérogènes. L'approche proposée pour la fusion de ces ontologies est composée de plusieurs étapes :

1. Construire la matrice d'ontologies où les ontologies du système A sont en première colonne et les ontologies du système B sont en première ligne.
2. Calculer la similarité sémantique entre les ontologies et donc remplir la matrice construite lors de la première étape. Pour cela, une mesure de similarité entre deux classes de deux ontologies différentes est proposée. Elle est calculée par la somme

pondérée de la similarité de trois parties de l'ontologie, similarité d'instances, similarité de définition et similarité de structure.

3. Construire le treillis de classes à partir de la matrice remplie lors de l'étape deux et en utilisant un algorithme de construction de treillis de classes (Fu & Nguifo, 2004; Snelting, 1994).

4. Dériver automatiquement l'ontologie globale à partir du treillis de classes construit lors de l'étape trois.

7.2.5. Algorithme de Mohsenzadeh et al.

L'algorithme de **(Mohsenzadeh, Shams, & Teshnehlab, 2005)** est construit comme suit : D'abord, pour chaque agent, une ontologie est créée, en langage naturel, par l'utilisateur non expert en ontologies indépendamment des autres agents et utilisateurs. Puis, les agents dont les ontologies sont du même domaine, sont exposés l'un à l'autre, échangent leurs ontologies, et décident de les fusionner. Ensuite, toutes les classes qui sont sémantiquement ou syntaxiquement similaires à celles dans l'ontologie de l'agent courant sont identifiées:

1. La similarité syntaxique est donnée par l'algorithme de la programmation dynamique basé sur la distance d'édition de Levenshtein.
2. La similarité sémantique est identifiée à partir des synsets du dictionnaire sémantique, WordNet.

Après cela, ces classes similaires (ainsi que toutes leurs relations) sont apprises par l'agent courant puis sont ajoutées à leur ontologie (par fusion avec leurs classes correspondantes). Et enfin, toutes les autres classes (non apprises par cet agent) ainsi que leurs relations sont identifiées, apprises et incorporées dans l'ontologie résultat de la fusion.

En revanche, l'agent explore l'ontologie qui lui est présentée par l'autre agent et identifie toutes les classes ainsi que les relations qu'il ne comprend pas (qu'il ne reconnait pas), il les apprend et les ajoute dans son ontologie. Finalement, pour relier les classes ajoutées à l'ontologie avec les classes qui existent déjà, wordNet est encore utilisé pour identifier les

types de relations qui doivent être utilisées telles que les relations d'hypernymie (super-classe), hyponymie (sous-classe) holonymie (est-partie-de), meronymie (a-pour), etc.

7.2.6. Algorithme de Kong et al.

L'algorithme de **(Kong, Hwang, & Kim, 2005)** est constitué de quatre étapes principales :

1. Alignement avec wordNet: C'est une étape de prétraitement pour mesurer la similarité de classes à travers différentes ontologies. WordNet est alors utilisé pour obtenir des valeurs plus riches que les valeurs déjà présentées dans l'ontologie.
2. Sélection de classes: Il s'agit de déterminer l'ensemble de classes sur lesquelles on va appliquer les mesures de similarité. Lors de cette étape, il est possible de restreindre la sélection sur un ensemble spécifique de candidats au processus de mesure de similarité (les paires de classes $\{(e, f)/ e \in O_1, f \in O_2\}$) et ignorer les autres.
3. Calcul de similarité: Pour déterminer les valeurs de similarité entre les paires de classes (e, f), candidates à l'alignement en se basant sur leurs définitions dans les ontologies O_1 et O_2, respectivement, d'abord, l'égalité entre classes est déterminée à base de wordNet. Ensuite, les valeurs de similarité sont calculées à travers les équations de Jaccard sim(e, f) et MSP(e, f) (Most Specific Parent). Finalement, les valeurs du PTR ainsi que la profondeur (depth) des classes dans wordNet sont utilisées pour déterminer les relations de sous-classe.
4. Reconstruction de la hiérarchie : En utilisant les valeurs calculées lors de l'étape précédente, les deux ontologies sont fusionnées et la nouvelle hiérarchie est reconstruite.

- **Comparaison des travaux**

Dans ce qui suit, nous présentons un résumé récapitulatif des principales différences entre les algorithmes de fusion d'ontologies complètement automatiques, décrits ci-dessus, selon un certains nombre de critères comme indiqué par la table 2.2 :

Table 2.2. Comparaison des travaux de la deuxième catégorie.

Propriétés	(Robin & Uma, 2010)	(Maiz et al., 2010)	(Cho, Kim, et al., 2006)	(Li et al., 2009)	(Mohsenzadeh et al., 2005)	(Kong et al., 2005)
Cadre de travail	Indépendant	Indépendant	Indépendant	Indépendant	Indépendant	Indépendant
Langage de représentation	OWL	Taxonomies	Taxonomies	Taxonomies	Taxonomies	Taxonomies
Langage de mapping	Mesure de similarité +	Mesure de similarité +	Mesure de similarité	Mesure de similarité	Mesure de similarité	Mesure de similarité

		fonction d'heuristiques	relations fils-père				
Ressources externes		WordNet	Aucun	WordNet	Aucun	WordNet	WordNet
Analyse lexicale		Oui	Oui	Non	Non	Oui	Non
Analyse sémantique		Oui	Oui	Oui	Oui	Oui	Oui
Niveau d' alignement	Classes	Oui	Oui	Oui	Oui	Oui	Oui
	Propriétés	Oui	Non	Non	Non	Non	Non
	Instances	Oui	Non	Non	Oui	Non	Oui
	Structure	Oui	Oui	Oui	Oui	Non	Non
Entrées		Deux ontologies	Plusieurs (≥2) ontologies	Deux ontologies	Plusieurs (≥2) ontologies	Plusieurs (≥2) ontologies	Deux ontologies
Sorties		Ontologie fusionnée	Ontologie fusionnée	Ontologie fusionnée	Ontologie fusionnée	Ontologie fusionnée	Ontologie fusionnée
Architecture de mapping		Point à point Bottom-up	Ce n'est pas applicable	Techniques horizontale et verticale	Point à point Top-down	Point à point Top-down	Point à point Top-down
Algorithme/ API utilisé		*	Algorithme de classification hiérarchique + inférence	*	Algorithme de similarité d'ontologies + algorithme de construction de treillis de classes	Algorithme de programmation dynamique	*
Apprentissage automatique		Non	Non	Non	Non	Non	Non
Développé à		Université Anna, Jerusalem, India, 2010	Lab ERIC, université lumière, France, 2010	Université Che Ju, Chosum, Korea, 2010	Université Hubei, Wuhan, China, 2009	Université KNT, Islamic Azad, Iran, 2005	Université Chosun, Ersity, Korea, 2005

* : Inconnu

7.3. Troisième catégorie : Algorithmes participant à la campagne OAEI'2015

7.3.1. Yam++ (not) Yet Another Matcher

Yam++ (Duyhoa & Bellahsene, 2013) est un système flexible (avec mono-configuration) d'alignement d'ontologies qui découvre les correspondances sémantiques entre classes, propriétés de type de données et propriétés d'objets. Après l'importation des ontologies sources, les informations des entités ontologiques sont indexées par les algorithmes d'indexation d'annotations par indexation de la structure et indexation du contexte. Ensuite, toutes les paires d'entités fortement similaires sont filtrées par un module de pré-filtration de candidats. En plus, le module de calcul de similarité extrait les mappings au niveau élément, en combinant les similarités obtenues au niveau terminologique (entre les

annotations des entités), au niveau instance (en mesurant le taux d'instances partagées entre classes d'ontologies sources) et au niveau contexte qui compare les profils contextuels des entités ontologiques. Le module d'alignement terminologique découvre les mappings entre les entités ontologiques en appliquant une technique de classification binaire sur un ensemble de données d'apprentissage appelé « Gold Standards ». Les mappings ainsi obtenus sont enrichis par les mappings au niveau structure qui sont obtenus en propageant les mappings initiaux à travers les structures des ontologies sources. Le module de post-filtration combine et sélectionne les mappings potentiels à partir des mappings obtenus au niveau élément et ceux obtenus au niveau structure. Ces derniers sont finalement vérifiés et raffinés en utilisant quelques modèles de détection de conflits sémantiques.

L'inconvénient de cette approche est qu'à chaque fois, elle demande l'intervention de l'utilisateur humain pour sélectionner le classifieur approprié ou utiliser un classifieur par défaut sur les mappings d'une base de connaissances volumineuse.

7.3.2. DKP-AOM (Disjoint Knowledge Preservation-based Automatic Ontology Merging)

DKP-AOM (Fahad, 2015) cherche les mappings sous les partitions de connaissances disjointes, pour réduire l'espace de recherche ainsi que le temps d'exécution. Pour fusionner deux ontologies sources, DKP-AOM suit une méthodologie composée de cinq étapes :

- D'abord, il génère les modèles intermédiaires des ontologies sources (les graphes OWL-DL) pour préparer et faciliter leurs exploration et manipulation et applique un ensemble d'opérations de pré-traitement sur les noms ainsi que les URIs des classes .
- Deuxièmement, en utilisant les graphes OWL-DL, précédemment générés, le composant « Match-Manager » effectue le travail du premier niveau pour détecter les mappings initiaux basés sur l'analyse lexicale, de synonyme et axiomatique entre les classes ainsi que leurs propriétés. Pour ce faire, d'abord, il construit l'espace de recherche à base de l'analyse des axiomes de disjonction présentés dans les ontologies sources, pour trouver les correspondances entre elles.

- Troisièmement, le module « Consistency Checker » exécute différents détecteurs de conflits (d'inconsistance, d'incomplétude et de redondances) pour valider la liste des mappings initiaux. Une fois que les mappings initiaux sont validés, le module Consistency Checker les diffuse vers le module « Reasoner ».
- Quatrièmement, le module Reasoner combine les mesures de similarités obtenues, résout les conflits et fusionne les mappings pour générer l'ontologie fusionnée globale.
- Finalement, l'ontologie fusionnée obtenue est compilée pour détecter et résoudre les conflits d'inconsistances, d'incomplétude et de redondances. Cependant, DKP-AOM souffre encore de certaines lacunes, tel que le fait que l'espace de recherche des mappings entre entités ontologiques est basé sur l'analyse des axiomes de disjonction, alors que cette information n'est pas toujours représentée dans les ontologies sources.

7.3.3. LogMap (**Log**ic-based **Map**ping)

LogMap (Jiménez-Ruiz and Grau, 2011, Jiménez-Ruiz et al., 2015) est un système évolutif d'alignement d'ontologies sources basé sur la logique propositionnelle de Horn. Il est basé sur deux principes majeurs : (1) Le principe de consistance (les mappings obtenus ne doivent pas mener à des classes inconsistantes dans l'ontologie intégrée). (2) Le principe de Localité (Les mappings doivent aligner les entités ayant des voisins similaires). Aussi, il supporte l'interaction, en temps réel, avec l'utilisateur humain durant le processus d'alignement et applique des techniques de raisonnement et de réparation pour minimiser le nombre de mappings logiquement, sauvegardées et indexées pour calculer les mappings initiaux en utilisant la technique d'indice inversé (inverted index technique). Ensuite, les ontologies sources sont modularisées en petits modules ayant des propriétés sémantiques compréhensibles. Ensuite, les modules obtenus ainsi que leurs mappings candidats sont codés en utilisant la logique propositionnelle de Horn. LogMap implémente l'algorithme classique de « Dowling-Gallier » (Dowling and Gallier, 1984) pour vérifier la satisfaction des propositions de Horn pour détecter les classes non satisfiables. Ensuite, il étend le même algorithme de « Dowling-Gallier» pour implémenter un algorithme de réparation (de scalabilité croissante) en détectant et éliminant tous les mappings qui peuvent introduire des inconsistances logiques.

Finalement, les clauses de Horn qui représentent les modules d'ontologies sources ainsi que leurs mappings sont sémantiquement indexées. L'indexation sémantique complète l'utilisation de la logique propositionnelle de Horn pour détecter et réparer les classes non satisfiables. LogMap reçoit en entrée deux ontologies sources et génère en sortie un ensemble de mappings de type [1-1] entre les classes et les propriétés. Mais, il n'implémente aucune technique spécifique pour aligner les instances. L'inconvénient de cette méthode est son inefficacité lors de l'alignement des ontologies dont les noms de classes et de propriétés sont cryptés ou éliminés (pour des raisons de sécurité, par exemple) car il n'utilise pas d'autres descripteurs tels que les instances ou les commentaires (ou descriptions).

7.3.4. XMap++ (eXtensible Mapping)

XMap++ (Djeddi and Khadir, 2013) est un outil flexible (avec mono-configuration) d'alignement automatique d'ontologies qui calcule les similarités lexicales (sur des chaînes de caractères), linguistiques (sémantique) et structurelles entre les classes et leurs propriétés (Data Type Properties et Object Properties). Pour contourner les problèmes des ambigüités lexicales, la similarité entre les entités des ontologies différentes est évaluée non pas seulement sur la base des sémantiques reflétées par les noms des entités mais également, en tenant compte de leur contexte (scope). En plus, l'algorithme XMap++ évalue les relations sémantiques entre deux entités ontologiques en tenant compte des types de cardinalités (e.g. *OWLCardinality, OWLAllValuesFrom, OWLSomeValuesFrom, Same_as, OWLMaxCardinality or Kind_of*) en plus des valeurs entre leurs propriétés (e.g. *OWLMinCardinality ≥1*). Les modules d'alignement effectuent les calculs des similarités tel que chaque entité de l'ontologie source est comparée avec toutes les entités (de même type) de l'ontologie cible, ce que produit trois matrices de similarités qui contiennent une valeur de similarité pour chaque paire d'entités. Ensuite, XMap++ applique la technique des réseaux de neurones artificiels pour apprendre et générer les poids (coefficients) lors de l'agrégation de différentes mesures de similarités. L'opérateur d'agrégation est utilisé pour combiner les différentes matrices de similarités calculées par les différents modules d'alignement pour obtenir une seule matrice de similarités combinées de taille n*m, où n

est le nombre d'éléments dans l'ontologie source et m est le nombre d'éléments dans l'ontologie cible. La liste finale des mappings est alors extraite de cette matrice en sélectionnant seulement les meilleurs mappings selon un seuil critique donné en plus de la cardinalité des correspondances (1-1). A l'instar de LogMap, XMap++ est inefficace lors de l'alignement des ontologies où les noms des classes et des propriétés sont éliminés parce qu'il n'utilise pas d'autre descripteurs tels que les instances ou les commentaires (descriptions).

- **Comparaison des travaux**

Dans ce qui suit, nous présentons un résumé récapitulatif des principales différences entre les algorithmes d'alignement et/ou de fusion d'ontologies complètement automatiques, décrits ci-dessus, selon un certains nombre de critères comme indiqué par la table 2.3:

Table 2.3. Comparaison des travaux de la troisième catégorie.

Propriétés		Yam++	DKP-AOM	LogMap	XMap++
Automation		Semi-Automatique	Automatique	Semi-Automatique	Automatique
Opération		Alignement	Fusion	Alignement	Alignement
Cadre de travail		Indépendant	Indépendant	Indépendant	Protégé2000
Langage de représentation		OWL	OWL	OWL	OWL
Langage de mapping		Mesure de similarité + Propagation de similarité	Mesure de similarité	Mesure de similarité	Mesure de similarité + Réseaux de neurones artificiels
Ressources externes		WordNet +Microsoft Bing	WordNet	WordNet	WordNet
Analyse lexicale		Oui	Oui	Oui	Oui
Analyse sémantique		Oui	Oui	Oui	Oui
Niveau d'alignement	Classes	Oui	Oui	Oui	Oui
	Propriétés	Oui	Oui	Oui	Oui
	Instances	Oui	Non	Oui	Non
	Structure	Oui	Oui	Oui	Oui
Entrées		Deux ontologies	Deux ontologies	Deux ontologies	Deux ontologies

Sorties	Liste de mappings	Ontologie fusionnée	Liste de mappings	Liste de mappings
Architecture de mapping	Point à point Top-Down	Top-Down Left-most	Top-Down	Point à point Top-down
Algorithme/ API utilisé	OWL-Api Top-K algorithm	OWL-DL Api + Jena + MorphAdorner Api + SimMetrics Api	Dowling-Gallier	OWL-DL Api + Jena +weka
Apprentissage automatique	Oui	Non	Non	Oui
Développé à	Lab. LIRMM. Université Montpellier, France, 2013	Lab. CERRAL, université lumière, France, 2015	Université d'Oxford, UK, 2015	Université Annaba, Algérie, 2015

8. Conclusion

L'émergence de la technique de fusion d'ontologies est un moyen par lequel on surmonte les restrictions et les spécificités des informations et des connaissances lorsqu'il s'agit d'une application qui touche deux (ou plus) domaines similaires. Dans ce chapitre, nous avons présenté un état de l'art sur la fusion d'ontologies. D'abord, nous avons détaillé le processus de fusion d'ontologies. Ensuite, nous avons décrit le processus de découverte de correspondances suivi par la présentation des critères d'évaluation d'un processus de fusion d'ontologies. Après cela, nous avons présenté quelques applications de la technique de fusion d'ontologies suivies par une description des principaux travaux d'alignement et de fusion, automatiques et semi automatiques qui existent dans la littérature. A ce stade, nous avons constaté que la plupart de ces travaux sont basés sur le calcul de similarité linguistique entre chaines de caractères pour détecter les classes similaires des ontologies sources, ce qui n'évitent pas les limites des techniques basées sur le TLN car elles sont toutes basées sur des dictionnaires ou des BDD lexicales. Pour confronter ce problème, dans la deuxième partie de la thèse, nous allons intégrer la technique de classification basée sur les arbres de décision pour détecter les classes similaires indépendamment de tous dictionnaires ou BDD lexicales. En outre, la plupart des systèmes existant sont basés sur la comparaison des noms des entités ontologiques,

ceci peut rater énormément de mappings intéressants, tels que le mapping (*writtenBy* Vs *reviewWrittenBy*).

Pour combler ces lacunes, nous allons intégrer également, un module d'alignement basé sur l'analyse structurelle des ontologies sources qui diffuse l'ensemble des mappings initiaux à travers les structures des ontologies sources (pour obtenir plus de mappings) en utilisant quelques heuristiques spécifiques.

Chapitre 3 . Arbres de décision.

1. Introduction

Dans beaucoup de domaines d'application, il est essentiel d'exploiter des techniques de classification compréhensibles par l'utilisateur.

Les arbres de décision sont l'une des méthodes d'apprentissage automatique symbolique les plus connues dans le domaine de l'intelligence artificielle. Ils sont parmi les techniques de classification les plus anciennes et les plus intuitives. Leur modélisation repose sur une représentation (structuration) graphique d'un ensemble de règles, qui les rendent aisément interprétables, rapides, apparemment simples, et donnant souvent des résultats convenables. C'est pourquoi cette approche a connu un vif succès dans différents domaines comme la représentation de connaissances, la fouille de données, ou encore les systèmes d'expertise (médicale, financière,… etc.) ainsi que l'aide au diagnostic médical où le médecin doit pouvoir interpréter les raisons du diagnostic.

Dans ce chapitre, nous présentons la technique des arbres de décision, tout en décrivant ces concepts inhérents et expliquant son processus d'apprentissage. Enfin, nous introduisons le contexte dans lequel nous allons utiliser la technique de classification basée-arbre de décision.

2. Concepts de base

2.1. Définition

Un arbre de décision (decision tree) est un enchaînement hiérarchique de règles logiques, sa structure est construite grâce à des méthodes d'apprentissage par induction à partir d'exemples, en cherchant à chaque niveau, l'attribut le plus discriminant pour classifier un exemple. L'arbre ainsi obtenu représente une fonction (qui a une traduction immédiate en terme de règles de décision mutuellement exclusives) qui fait la classification d'exemples, en s'appuyant sur les connaissances induites à partir d'une base d'apprentissage. En raison de cela, ils sont aussi appelés arbres d'induction *(Induction decision tree*s) (Osório, 1998).

2.2. Principe

En représentation de connaissances, la structure de l'arbre peut être élaborée manuellement pour opérer une modélisation explicite, voir structurelle des connaissances. Lorsqu'un apprentissage automatique est effectué, les informations les plus pertinentes, permettant de partitionner l'espace de représentation en régions de décision représentatives d'une classe, sont extraites de la base d'apprentissage. Le partitionnement final est obtenu en utilisant une stratégie de type « *diviser pour régner* » afin de ne pas considérer le problème de manière globale, mais, au contraire pour le subdiviser en sous-problèmes plus faciles à traiter.

Pour cela, des partitionnements récursifs de l'espace (de la base) sont faits selon des conditions sur les valeurs des attributs (catégories d'une variable qualitative ou intervalles continus de valeurs pour une variable quantitative). Le choix des conditions est effectué en fonction de leur pouvoir de discrimination, ce qui permet de les organiser hiérarchiquement, rendant ainsi le processus de décision plus fiable et plus robuste.

Une particularité de ce principe de modélisation est qu'il est possible de mettre « à plat » la structure arborescente pour pouvoir travailler directement sur une base de règles de décision. Ce formalisme est plus lisible et plus facile à manipuler que la structure d'arbre en elle-même.

Nous notons que l'objectif des arbres de décision est de mettre en relation un ensemble de connaissances relatives aux propriétés des formes dans l'espace de représentation de façon à pouvoir discriminer les classes auxquelles elles appartiennent. Ces connaissances se présentent sous la forme de conditions sur la valeur des attributs et leurs relations sont décrites par une arborescence facilement interprétable.

2.3. Représentation formelle

De manière formelle, un arbre de décision est un graphe orienté sans cycle $G = (N, A)$ composé d'un ensemble de *noeuds* (sommets) N et un ensemble d'*arcs* A. Les noeuds terminaux, appelés *feuilles*. Un *chemin* (ou branche) dans l'arbre est une suite d'arcs qui relie deux sommets. Un noeud non terminal N_{Id} est étiqueté par un des attributs Fk, k =1,…, n de l'espace de représentation. Les arcs sont, quant à eux, étiquetés par une condition Fkl portant sur les valeurs

de *Fk* et qui permet de passer du noeud père *Id* au nœud fils *Id.l* si elle est satisfaite. Cette condition représente en général une modalité de l'attribut ou un ensemble de valeurs. Enfin, les feuilles sont étiquetées par une classe $\omega i \in S$ représentant la décision qui doit être prise si une forme satisfait l'ensemble des conditions le long du chemin allant de la racine à la feuille (Ragot, 2003).

La figure 3.1 présente un exemple partiel et simplifié d'arbre de décision pour la discrimination de lettres latines. Les attributs de l'espace de représentation portent ici sur les propriétés morphologiques des tracés et les classes sont les lettres de l'alphabet latin (Ragot, 2003).

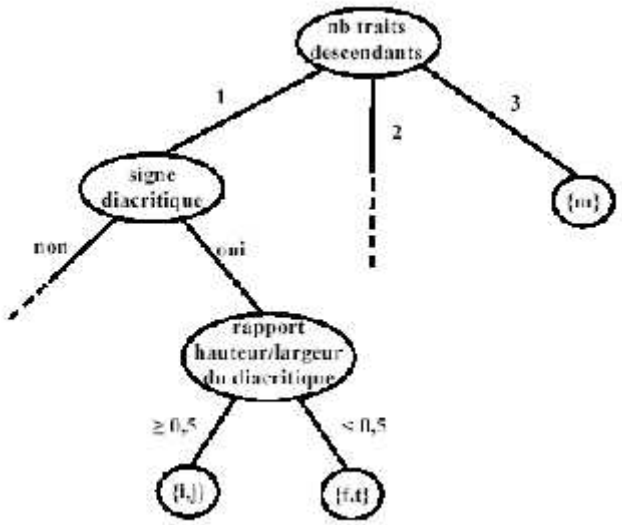


Figure 3-1. Exemple d'arbre de décision (partiel) pour la discrimination de caractères latins

2.4. Traduction de la position des nœuds

Chaque nœud N_{Id} de l'arbre possède un identifiant que nous notons *Id* d'une façon générale. Cet identifiant est en fait une séquence d'indices repérant la position du noeud dans l'arbre. Chaque arc $(N_{Id}, N_{Id.l})$ est orienté et relie un noeud *père* N_{Id} et un noeud *fils* $N_{Id.l}$ où *Id.l* est la concaténation (.) de l'identifiant *Id* du *père* avec l'indice *l* du *fils* (son numéro parmi l'ensemble des *fils* du noeud *père*). Chaque noeud possède exactement un père à l'exception de la racine de l'arbre qui n'en possède aucun. Son identifiant est 1 et elle est notée *N1*.

Si on considère l'arbre de la figure 3.1, la traduction des positions de ces nœuds sera :

N1 : étiqueté par le test : nb traits descendants.

N1.1 : étiqueté par le test : Signe diacritique.

N1.1.2 : étiqueté par le test : Rapport hauteur/largeur du diacritique

N1.1.2.1 : étiqueté par : {i, j}.

N1.1.2.2 : étiqueté par : {f, t}.

N1.3 : étiqueté par : {m}

2.5. Extraction de règles

Grâce à la structuration des connaissances sous cette forme, l'interprétation, sous forme de règles est rendue relativement aisée et améliore considérablement la lisibilité et la flexibilité. Ainsi sur l'exemple de la figure 3.1, on trouve :

- **Si** (*un caractère possède trois traits descendants*) **alors** *c'est un* **{m}.**
- **Si** *(un caractère ne possède qu'un trait descendant et qu'il possède un signe diacritique dont le rapport hauteur/largeur est supérieur ou égal à 0.5)* **alors** *il s'agit d'un* **{i}** *ou d'un* **{j}.**

Notons que cet exemple n'est que partiel et qu'il ne permet pas de faire une discrimination complète des classes (les feuilles ne sont pas étiquetées par une seule classe). Pour pouvoir discriminer le 'i' du 'j' et le 'f' du 't', il faudrait raffiner l'arbre en y incorporant d'autres connaissances discriminantes.

Ce type de structure a été utilisé dans de nombreux domaines applicatifs où son interprétabilité ainsi que son habileté à manipuler des attributs à la fois numériques et symboliques en a fait une approche privilégiée par rapport à d'autres approches de classification. Cependant, dans le cadre de la reconnaissance de formes sa capacité à gérer efficacement les données numériques reste limitée. En effet comme la plupart des données réelles manipulées sont sujettes aux imprécisions et au bruit, l'utilisation de conditions strictes comme « >=0.5 » provoque parfois des comportements indésirables et forme une source d'erreurs (Ragot, 2003).

2.6. Représentation textuelle

L'arbre de décision peut être affiché sous forme textuelle en effectuant un décalage vers la droite pour chaque « niveau » de l'arbre, par exemple l'arbre de la figure 3.1 devra être affiché textuellement sous la forme suivante :

| *Nb traits descendants = 1 :*

| | *Signe diacritique = oui :*

| | | *rapport hauteur_ largeur >= 0.5 : {i, j}*

| *rapport hauteur_ largeur < 0.5 : {f, t}*

| *Nb traits descendants = 3: {m}*

2.7. Avantages

Les arbres de décision présentent de nombreux avantages :

- Leur lisibilité, car ils peuvent être traduits sous forme de règles
- Plusieurs domaines d'application ont montré l'intérêt de l'utilisation des arbres de décision, on prend à titre d'exemples : la prédiction, la reconnaissance et la classification des connaissances…
- L'expressivité des arbres de décision leur permet l'approximation de fonctions à valeurs discrètes et les rend capables d'apprendre des expressions disjonctives.
- Le nombre moyen de tests à effectuer sur une forme peut être réduit (si les *d* attributs sont tous binaires, ce nombre est limité par *d*).
- La structure de décision est globale à toutes les classes: on n'a pas de problème pour traiter directement C classes.
- Il n'est pas nécessaire de tester tous les attributs de chaque forme à chaque nœud, dans la plupart des cas pratiques, on se limite même à un seul (sélecteur).
- Il existe actuellement des solutions pour les problèmes d'apprentissage par les arbres de décision. Par exemple, pour éviter le sur-apprentissage, il est possible d'arrêter la croissance de l'arbre quand la division des données n'est plus statistiquement significative ou de générer l'arbre entier, puis élaguer. Comme il est possible de générer plusieurs arbres et de sélectionner le meilleur d'entre eux en mesurant les performances sur un ensemble distinct de données de validation, en mesurant les performances sur l'ensemble d'apprentissage et en effectuant des tests statistiques, ou encore en minimisant la taille de l'arbre ainsi que ses erreurs de classification.

2.8. Inconvénients

Comme toute approche d'apprentissage, les arbres de décision présentent aussi certains inconvénients

- La difficulté de manipulation de variables quantitatives (valeurs continues). La discrétisation des données reste une solution à considérer, mais elle ne résout pas tous les problèmes de traitement d'informations non symboliques.

- Le choix des attributs à considérer dans les divisions (nœuds de l'arbre) ne garantit pas une solution toujours parfaite. De plus, la méthode la plus utilisée, fondée sur la théorie de l'information de Shannon, impose de fortes contraintes aux méthodes incrémentales (on est obligé de tout reconstruire).
- Les arbres de décision n'utilisent pas de connaissances théoriques disponibles sur le problème posé. Ils exploitent uniquement des connaissances empiriques.
- Les arbres de décision ont tendance à être instables, c'est à dire que de petites fluctuations dans la base d'exemples utilisée peuvent considérablement modifier la topologie de l'arborescence.

3. Apprentissage par arbres de décision

Dans ce paragraphe, nous décrivons un processus d'apprentissage classique permettant d'élaborer automatiquement un arbre de décision c'est- à dire d'extraire les connaissances discriminantes et de les structurer sous forme d'une arborescence.

Le processus d'apprentissage classique peut se faire de différentes manières selon divers algorithmes caractérisés par leurs propres particularités. Parmi les algorithmes les plus connus et les plus utilisés on peut citer l'algorithme CART (Classification and Regression Trees), de Breiman et al. (1984) et l'algorithme ID3 de Quinlan (1986) (Ragot, 2003) qui sont à l'origine de beaucoup d'autres algorithmes de construction d'arbres de décision classiques. La présentation qui suit s'inspire des approches descendantes et plus particulièrement de l'algorithme C4.5 qui est dérivé lui même de l'ID3.

La construction de l'arbre s'effectue à partir d'une base d'apprentissage B_{app} initiale. L'objectif est d'extraire de cette base les connaissances pertinentes permettant de partitionner la base d'apprentissage en sous-ensembles d'individus représentatifs d'une unique classe opérant ainsi une discrimination des classes. Les partitionnements s'effectuent de manière récursive sur une base d'exemples B ($B = B_{app}$ au départ) à partir de conditions sur les valeurs des attributs F_k, $k=1,\ldots, n$ de l'espace de représentation des formes. Ces attributs prennent un ensemble de valeurs (ou modalités), F_{kl}, $l= 1.\ldots, L_k$. L'algorithme d'apprentissage se résume alors de la manière suivante (Ragot, 2003):

INITIALISATION :
B = Bapp.
CONSTRUIRE ARBRE *(B)*
DEBUT
SI le critère d'arrêt est vérifié
ALORS créer une feuille
SINON
Choisir un attribut Fk et créer un nœud
Partitionner *B* en L_k sous- ensembles B_{kl}, $l = 1,\ldots, L_k$
Pour $l = 1, \ldots, L_k$ CONSTRUIRE- ARBRE (B_{kl})
FINSI
FIN

Cet algorithme récursif est caractérisé par quatre paramètres fondamentaux qui permettent de distinguer les différentes approches :

- Le choix d'un attribut pour partitionner B.
- Une stratégie de partitionnement.
- Un critère d'arrêt.
- La création d'une feuille.

3.1. Le choix d'un attribut pour partitionner

Lors de la construction d'un noeud non terminal, il faut déterminer l'attribut qui discrimine le mieux la base courante B d'individus. Rappelons que les connaissances discriminantes ne sont pas toutes pertinentes puisqu'elles sont caractérisées par leur pouvoir de discrimination.

De plus, cette pertinence est fortement dépendante du contexte dans lequel elle est évaluée. Choisir les attributs les plus discriminants localement à chaque nœud permet donc d'obtenir une modélisation plus robuste mais aussi plus compacte et lisible.

Pour évaluer la pertinence d'un attribut relativement à la distribution en classes d'un ensemble d'individus on utilise le plus souvent une mesure de discrimination. De nombreux travaux ont été effectués sur l'élaboration de ces mesures et les propriétés qu'elles doivent posséder (Ragot, 2003), mais la mesure la plus utilisée est probablement ***l'entropie*** de Shannon (proposée en 1948) qui repose sur les principes de la théorie de l'information.

Rappelons tout d'abord que $S = \{\omega_1,\ldots.. \omega_i,\ldots, \omega_s\}$ est l'ensemble des classes et que B désigne un ensemble de N individus à partitionner au niveau d'un nœud. Parmi ces échantillons, Ni appartiennent à la classe ω_i et la probabilité associée est :

$$p(w_i) = \frac{N_i}{N}$$

L'information relative à la distribution en S classes sur B s'écrit:

$$HB(S) = -\sum_{i=1..S} p(w_i)\log_2 p(w_i) \qquad (1)$$

Ainsi HB(S) sera maximale si $p_1 = p_2 = 0.5$ c'est à dire si les distributions sont fortement corrélées. Elle sera au contraire minimale si p_1 ou p_2 est nul.

Considérons maintenant un attribut F_k prenant les modalités F_{kl}, $l = 1,\ldots, L_k$.

Si cet attribut est choisi pour partitionner B, les N exemples sont partagés en L_k sous- ensembles B_{kl}, $l = 1,\ldots, L_k$ suivant la modalité à laquelle ils appartiennent. Les effectifs associés aux B_{kl} sont notés N_{kl}.

Parmi eux N^i_{kl} appartiennent à la classe ω_i. L'entropie associée à cette division relativement aux classes est appelée *entropie conditionnelle* :

$$HB(S \setminus F_k) = -\sum_{i=1..Lk} p(F_{kl}) \sum_{i=1..S} p(w_i \setminus F_{kl})\log_2 p(w_i \setminus F_{kl}) = \sum_{l=1..Lk} p(F_{kl}) HB_{kl}(S) \qquad (2)$$

Avec : $p(w_i \setminus F_{kl}) = \dfrac{N_{kl}}{N^i_{kl}}$

et : $p(F_{kl}) = \dfrac{N_{kl}}{N}$

Prendre l'attribut *Fk* le plus discriminant revient alors à choisir celui qui maximise **le gain d'information :**

$$GB(F_k) = HB(S) - HB(S \setminus F_k) \qquad \textbf{(3)}$$

Il existe beaucoup d'autres mesures pour choisir le meilleur attribut comme : la mesure d'impureté de Gini ou encore le critère de ¡². De nombreuses études empiriques convergent vers la même constatation, à savoir qu'aucune mesure n'est meilleure qu'une autre de manière significative, et que le choix de ce critère est nettement moins important pour l'efficacité de l'arbre de décision que d'autres paramètres comme le critère d'arrêt. Choisir l'une de ces mesures dépend du type de l'arbre que l'on souhaite construire. Par exemple le critère de Gini se révèle mal adapté pour mesurer l'impureté quand plusieurs classes doivent être séparées. De plus, ce critère tend à favoriser les attributs produisant une partition dans laquelle les sous-ensembles sont de taille et de pureté équivalentes. Au contraire, l'entropie favorise les partitions les plus

pures, ce qui peut aboutir à des arbres déséquilibrés. Certains auteurs ont aussi montré que cette mesure était biaisée en faveur des attributs possédant un grand nombre de modalités (Ragot, 2003).

3.2. La stratégie de partitionnement

La procédure de construction des arbres de décision vise à partitionner l'ensemble d'apprentissage jusqu'à obtenir des sous-ensembles d'individus appartenant tous à la même classe. On dit alors que ces sous-ensembles sont « purs ». Les partitionnements s'effectuent dans l'espace de représentation suivant les différentes modalités des attributs. Il existe alors différentes possibilités pour effectuer un partitionnement au niveau d'un nœud selon que les attributs soient nominaux ou continus.

a) Le cas nominal

Dans le cas des attributs binaires, le test consiste à descendre dans un sous arbre si le test sur l'attribut choisi vaut VRAI, dans l'autre quand il vaut FAUX.

Le cas où les attributs sont à valeurs discrètes se généralise facilement quand le test que l'on construit se réduit à opposer une valeur à toutes les autres, on est alors ramené au cas binaire. Par exemple, s'il existe un attribut couleur prenant ses valeurs dans l'ensemble *{bleu, rouge, vert}*, il est simple de l'éclater en trois attributs binaires, du type couleur-rouge, couleur-bleu couleur-vert qui est VRAI ou FAUX sur chaque donnée d'apprentissage. On se replace alors dans le cas exposé ci-dessus, avec la transformation d'un attribut nominal à k valeurs possibles en k attributs binaires que l'on traite indépendamment. C'est l'un des deux processus utilisés par l'algorithme CART (lorsque le nombre d'attributs est grand). La deuxième méthode, utilisée quand le nombre d'attributs est réduit, consiste à dériver un attribut pour chaque paire de modalités possible ({rouge, vert} {rouge, bleu} {vert, bleu} dans le cas de notre exemple).

Cette technique a l'inconvénient d'oublier la signification globale de l'attribut ; en effet, si couleur-rouge est VRAI pour un attribut, couleur-bleu est automatiquement FAUX, mais cette propriété n'apparaît plus explicitement dans les données. Une autre solution est alors de calculer directement l'information mutuelle entre les deux variables à valeurs discrètes que sont d'une part cet attribut et d'autre part l'ensemble des classes. Si celle-ci se révèle la meilleure pour tous les attributs, on crée alors un nœud non binaire dans l'arbre de décision (dans l'exemple précédent, le test de l'attribut « couleur » donne quatre réponses possibles). Le seul inconvénient

est qu'il faut gérer une structure de données plus complexe (Cornuéjols, Miclet, & Kodratoff, 2010).

b) Le cas continu :

Traiter un attribut continu peut paraître plus difficile, mais en pratique ce n'est pas fondamentalement différent, car le nombre de données d'apprentissage est fini, le nombre des valeurs que prend cet attribut sur les exemples est aussi fini. Mieux, ses valeurs sont ordonnées, contrairement au cas nominal. Le sélecteur consistera donc à comparer les valeurs à un seuil pour construire un nœud binaire.

Pour un attribut ***a*** continu, on procède alors ainsi : on trie les points d'apprentissage selon la valeur de cet attribut, puis on cherche le seuil s(a) qui minimise l'un des critères précédents.

Il est à noter que l'arbre de décision est le seul modèle permettant de gérer de manière homogène les attributs de natures variées, en particulier les mélanges continus et binaires (Cornuéjols, Miclet, & Kodratoff, 2010).

Les stratégies de partitionnement des attributs continues ont fait l'objet de plusieurs études, des chercheurs ont essayé de faire des partitionnements en utilisant plusieurs attributs à la fois pour pouvoir prendre en compte l'interaction entre eux, notamment du point de vue de leur pouvoir de discrimination en se basant sur leur combinaison linéaire, non linéaire ou en utilisant les réseaux de neurones. Les arbres de décision obtenus sont plus efficaces mais plus coûteux à apprendre et à construire (Ragot, 2003).

3.3. Le critère d'arrêt

Le critère d'arrêt est un point essentiel de la procédure de construction de l'arbre car il détermine en grande partie son efficacité tant au niveau de sa compacité que de ses capacités de généralisation. Une première idée simple consiste à grossir l'arbre (en réitérant la procédure CONSTRUIRE-ARBRE) tant que les nœuds ne sont pas purs, c'est à dire tant qu'ils contiennent des exemples de classes différentes. L'arbre de décision ainsi obtenu décrit exactement l'ensemble d'apprentissage et ne fait aucune erreur de classification sur cette base. Cette approche possède deux inconvénients majeurs : d'une part les capacités de généralisation de l'arbre de décision seront très souvent mauvaises conduisant à un sur-apprentissage, d'autre part, la taille de l'arbre risque de devenir très importante si les classes sont difficilement séparables. L'impact sur la taille de l'arbre dépend en général du nombre d'exemples dans la base

d'apprentissage, plus le nombre d'individus utilisé en apprentissage est important, plus l'arbre risque d'être de taille importante. Ce phénomène est accru par la présence de bruit dans les données. Outre les problèmes d'utilisation de ressources mémoire et CPU que cela suscite, un arbre trop grand est difficilement interprétable ce qui n'est pas souhaitable en général.

Pour limiter ces effets néfastes, d'autres critères d'arrêt sont introduits dans la procédure de construction. On distingue principalement deux approches: l'arrêt « précoce » et « l'élagage ».

La première approche agit pendant la construction de l'arbre tout comme le critère d'arrêt sur la pureté d'un nœud. Classiquement une feuille est construite lorsque le nombre d'échantillons de la base B d'un nœud est trop faible ou encore lorsque la pureté de ce sous-ensemble d'individus est jugée suffisante. Cette estimation peut se déduire directement de la mesure de discrimination utilisée pour les partitionnements. Cependant, ces mesures sont calculées localement à un nœud et sont souvent fortement dépendantes du nombre d'échantillons. Il est donc difficile d'établir un seuil unique qui puisse être valable pour tous les nœuds de l'arbre. La construction de l'arbre risquerait alors de s'arrêter prématurément pour certains nœuds ou, au contraire, tardivement pour d'autres. D'autres variantes ont été conçues pour pallier à ces problèmes comme par exemple une mesure d'impureté qui se calcule sur l'arbre complet.

Au lieu d'arrêter la construction de l'arbre de manière « précoce », certains chercheurs suggèrent de construire l'arbre de décision de taille maximale puis de réduire sa taille dans une seconde phase par une méthode d'élagage. Le principe général consiste à supprimer des nœuds ou des sous-arbres peu représentatifs, ou encore à les regrouper. De nombreuses techniques ont été étudiées montrant pour la plupart une efficacité réelle par rapport à l'arbre non élagué. Certaines de ces techniques procèdent en deux étapes : la première consiste à construire une série d'arbres de plus en plus petits à partir de la base d'apprentissage en supprimant les nœuds en fonction d'un critère basé sur le rapport entre le taux d'erreur de l'arbre sur la base d'apprentissage et sa taille *(cost complexity*). La seconde utilise une base de validation pour choisir l'arbre qui a le taux d'erreur le plus faible sur cette base.

3.4. La création d'une feuille

Une feuille diffère d'un nœud non terminal par le fait qu'elle marque l'arrêt de l'expansion d'une branche parce qu'un des critères d'arrêt a été satisfait. La feuille signifie alors que l'on peut (doit) être capable d'identifier la classe la plus représentative dans la sous base B local. Lorsque cette sous base est pure, il n'y a qu'une seule classe possible et c'est elle qui étiquette la feuille. C'est

le cas idéal, mais il est rarement satisfait en pratique. Dans tous les autres cas, la règle générale consiste à étiqueter la feuille avec la classe possédant le plus de représentants.

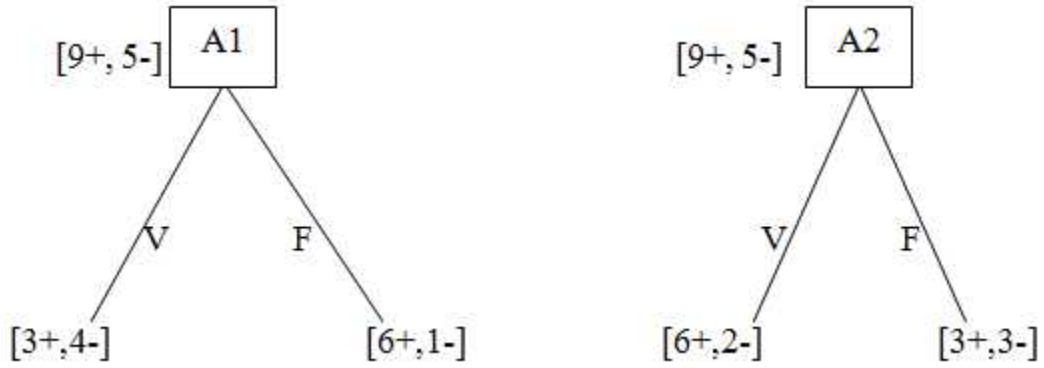
3.5. Exemple d'application

L'exemple suivant illustre les étapes de construction d'un arbre de décision pour l'ensemble d'exemples du tableau 3.1 (Mitchell, 1997).

Table 3.1. Ensemble d'exemples pour la construction d'un arbre de décision

	Ciel	Température	Humidité	Vent	Jouer
J1	Soleil	Chaud	Élevée	Faible	Non
J2	Soleil	Chaud	Élevée	Fort	Non
J3	Couvert	Chaud	Élevée	Faible	Oui
J4	Pluie	Doux	Élevée	Faible	Oui
J5	Pluie	Froid	Normale	Faible	Oui
J6	Pluie	Froid	Normale	Fort	Non
J7	Couvert	Froid	Normale	Fort	Oui
J8	Soleil	Doux	Élevée	Faible	Non
J9	Soleil	Froid	Normale	Faible	Oui
J10	Pluie	Doux	Normale	Faible	Oui
J11	Soleil	Doux	Normale	Fort	Oui
J12	Couvert	Doux	Élevée	Fort	Oui
J13	Couvert	Chaud	Normale	Faible	Oui
J14	Pluie	Doux	Élevée	Fort	Non

Choix de l'attribut



Entropie

$$Entropie\ (S) = -(p+)\log_2(p+) - (p-)\log_2(p-) \quad (4)$$

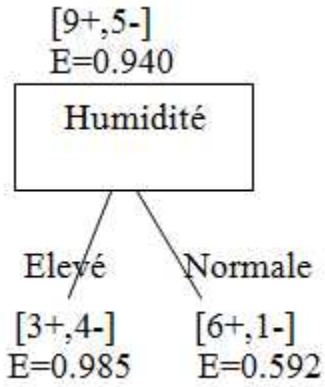
- S est un ensemble d'exemples
- p+ est la proportion d'exemples positifs
- p- est la proportion d'exemples négatifs
- L'entropie Mesure l'homogénéité des exemples

$$Entropie\ ([9+,5-]) = -(9/14)\log_2(9/14) - (5/14)\log_2(5/14) = 0.940$$

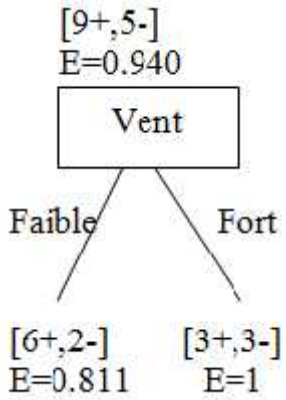
Gain d'information

Gain(S, A)=Réduction d'entropie due à un tri suivant les valeurs de A

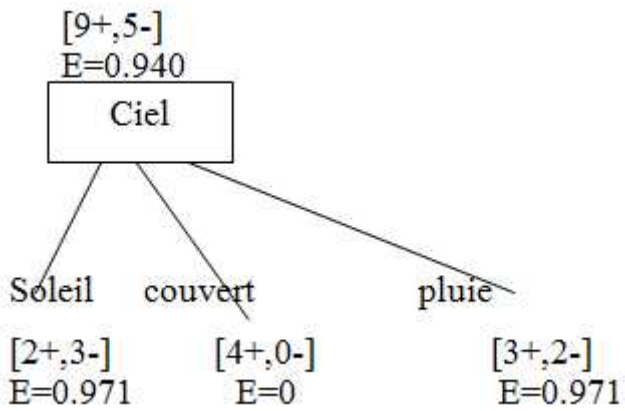
$$Gain\ (S, A) = Entropie\ (S) - \sum_{v \in valeur\ (A)} (|S_v|/|S|) * Entropie\ (S_v) \quad (5)$$



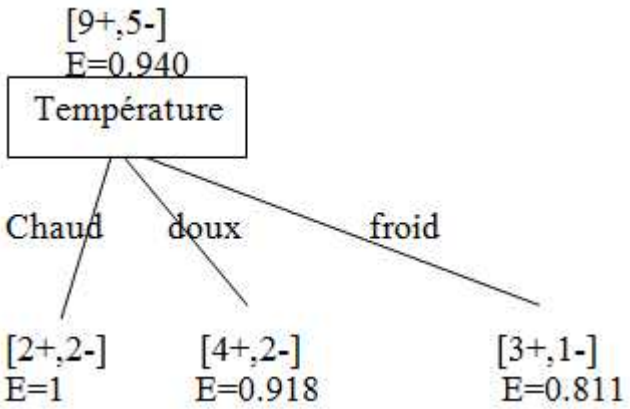
Gain(S,Humidité)=0.940-(7/14) *0.985-(7/14)*0.592=0.151



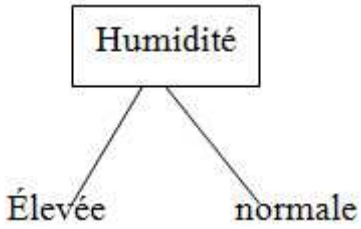
Gain (S, vent)=0.940-(8/14)*0.811- (6/14)*1=0.048



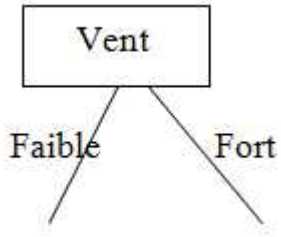
Gain(S,ciel)=0.940-(5/14)*0.971-(5/14)*0.971-0=0.246



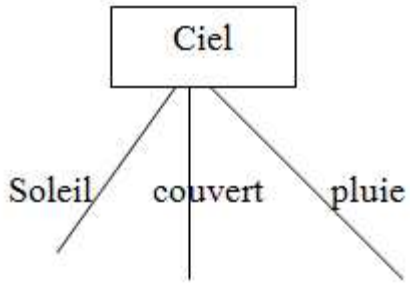
Gain(S, température)=0.940-(4/14)*1-(6/14)*0.918-(4/14)*0.811=0.029



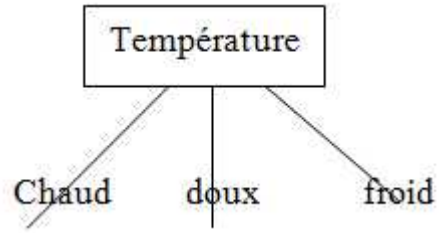
Gain(S, Humidité) =0.151



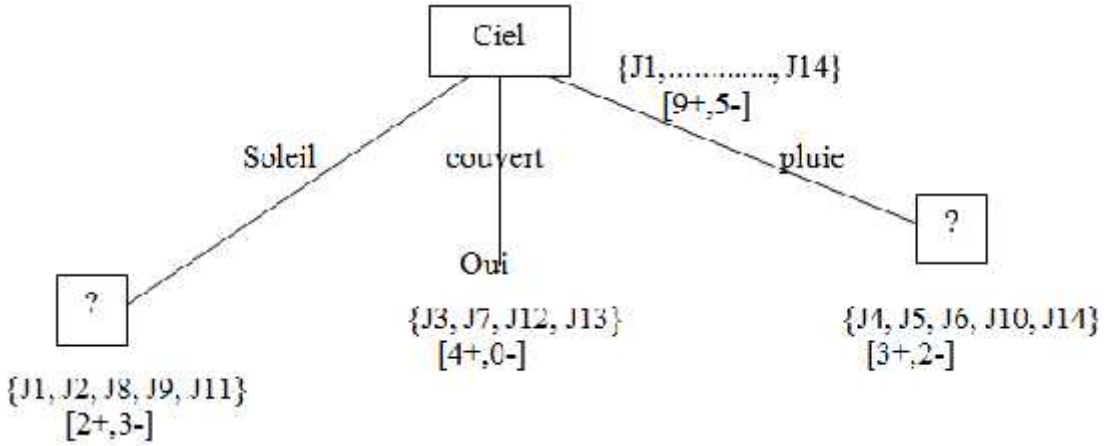
Gain(S, vent)=0.048



Gain(S, ciel)=0.246



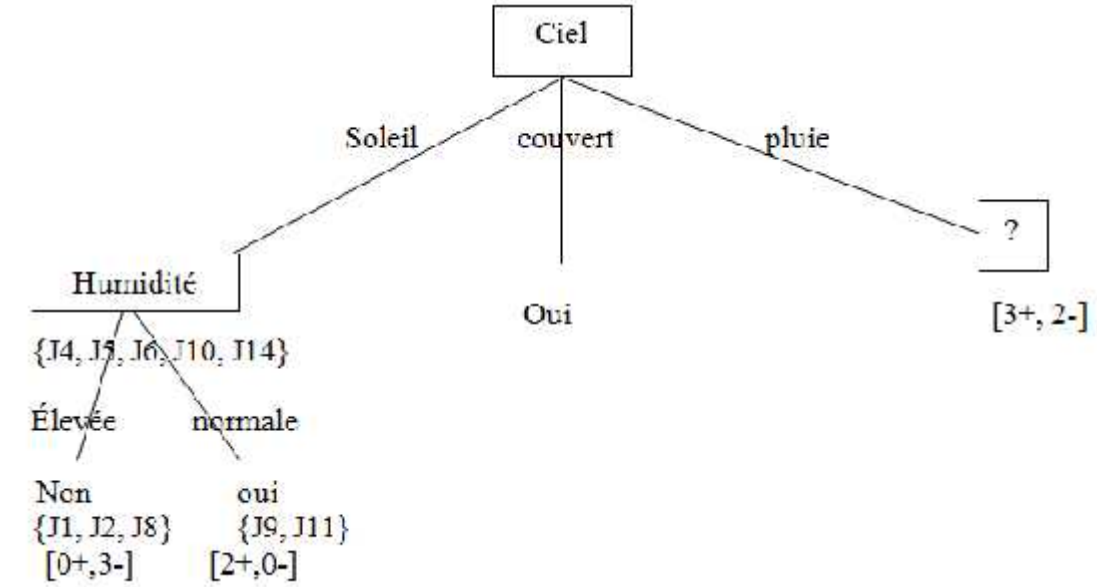
Gain(S, température)=0.029

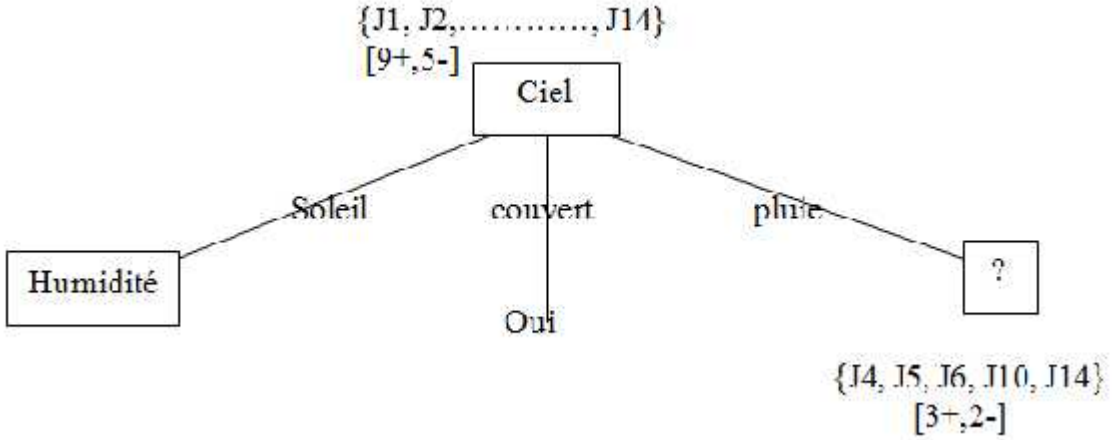


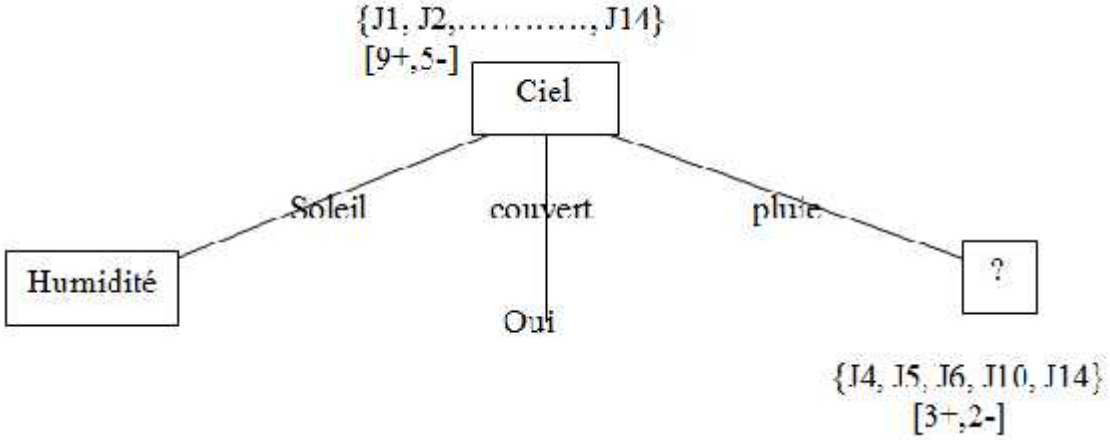
Gain (S soleil,Humidité)=0.970-(3/5)*0-(2/5)*0=0.970

Gain (S soleil,température)=0.970-(2/5)*0-(2/5)*1-(1/5)*0=0.570

Gain (S soleil, vent)= 0.970-(2/5)*1-(3/5)*0.918=0.019


Ciel
Soleil
couvert
pluie
?
Humidité
Oui
[3+, 2-]
{J4, J5, J6, J10, J14}
Élevée
normale
Non
{J1, J2, J8}
[0+,3-]
oui
{J9, J11}
[2+,0-]


{J1, J2,……, J14}
[9+,5-]
Ciel
Soleil
couvert
pluie
Humidité
Oui
?
{J4, J5, J6, J10, J14}
[3+,2-]

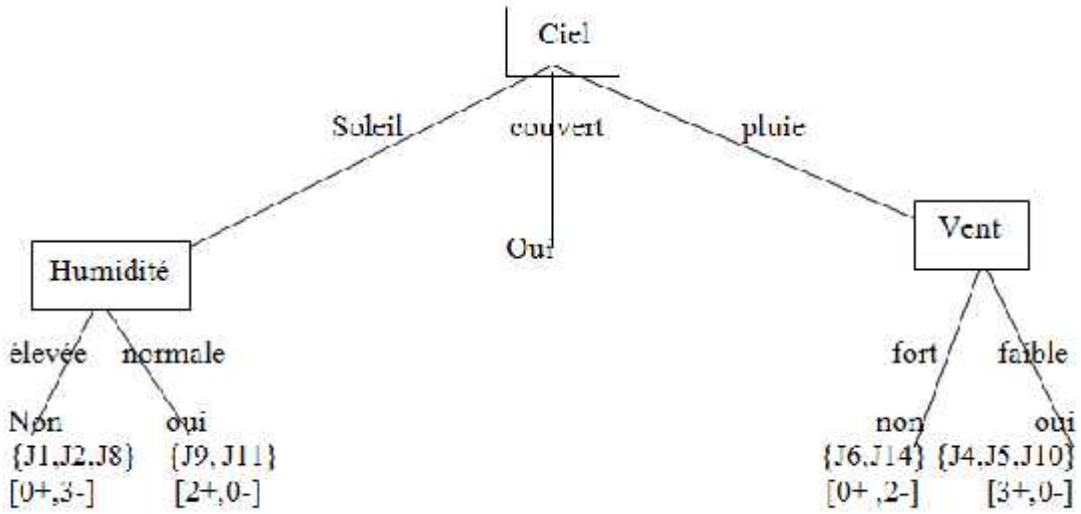

{J1, J2,……, J14}
[9+,5-]
Ciel
Soleil
couvert
pluie
Humidité
Oui
?
{J4, J5, J6, J10, J14}
[3+,2-]

Gain (S pluie,Humidité)=0.970-(2/5)*1-(3/5)*0.918=0.019

Gain (S pluie,température)=0.970-(0/5)-(3/5)*0.918-(2/5)*1=0.019

Gain (S pluie, vent)= 0.970-(2/5)*0-(3/5)*0=0.970

L'arbre de décision obtenu est alors :



4. Élagage des arbres de décision

La réitération de l'algorithme de construction jusqu'à son terme naturel résulte en un arbre de taille maximale T_{max} dont les feuilles sont pures, c'est-à-dire correspondant à la même classe. L'élagage consiste à minimiser la taille de l'arbre de décision en remplaçant un sous arbre par une feuille ou en regroupant plusieurs classes en une seule. Il peut se faire de deux manières: soit pendant la construction de l'arbre (pré-élagage), soit à posteriori, une fois que l'arbre aura été entièrement développé (post-élagage) (Cornuéjols, Miclet, & Kodratoff, 2010).

4.1. Le pré-élagage

Il est effectué au fur et à mesure de la construction de l'arbre de décision où on cesse de diviser un nœud quand la pureté des points qu'il domine est non pas parfaite mais suffisante (si l'entropie calculée au niveau de ce nœud est inférieure à certain seuil), on le considère alors comme une feuille et on lui attribue la classe en question. Cette méthode ne prend en compte qu'un critère local à la feuille examinée et peut de ce fait manquer un développement intéressant de l'arbre.

4.2. Le post-élagage

Une autre technique, plus valide théoriquement et plus efficace en pratique, consiste d'abord à construire l'arbre de décision complètement, puis chercher à le simplifier en l'élaguant progressivement en remontant des feuilles vers la racine. Pour juger quand il est bon d'arrêter d'élaguer l'arbre, on utilise un critère de qualité qui exprime souvent un compromis entre l'erreur commise par l'arbre et une mesure de complexité. L'erreur commise est mesurée grâce à un ensemble de validation. On supposera donc dans ce paragraphe que l'ensemble d'apprentissage est assez important pour être coupé en deux parties: l'une (ensemble d'apprentissage proprement dit) pour construire l'arbre de décision T_{max}, l'autre (ensemble de validation) pour choisir le meilleur parmi les élagages proposés. L'algorithme optimal consisterait à calculer le taux d'erreur de l'ensemble de validation sur tous les arbres qu'il est possible d'obtenir par élagage de T_{max}. Mais leur nombre croit très rapidement avec la taille de T_{max}, mesurée en nombre de nœuds. On utilise donc des solutions sous-optimales, dont la plus classique (un algorithme *glouton*) consiste à construire, sans retour en arrière, une séquence d'arbres par élagages successifs, en remontant des feuilles vers la racine. Cette séquence se note S= (T_{max},T_1,…,T_k,…..,T_n). T_n est l'arbre constitué d'une seule feuille comprenant les m points d'apprentissage. C'est donc l'arbre élagué au maximum. Pour passer de T_k à T_{k+1}, il faut transformer un nœud dans T_k en feuille. Pour savoir si cet élagage serait bénéfique, l'idée générale est de comparer le «coût» de l'arbre élagué et celui de l'arbre non élagué, et d'arrêter l'élagage quand le coût du premier dépasse le coût du second. Pour évaluer ce coût, plusieurs critères ont été proposés qui prennent tous en compte à la fois l'erreur commise par l'arbre et une mesure de sa complexité (Cornuéjols, Miclet, & Kodratoff, 2010).

Nous examinons ici le critère consistant à choisir le nœud v qui minimise sur l'ensemble des nœuds de Tk, la valeur suivante (qui représente un compromis entre la taille de l'arbre élagué et le taux d'erreur) :

$$W(Tk, v) = (MC_{éla}(v,k) - MC(v,k)) / (n(k) * (nt(v,k) - 1)) \qquad (6)$$

Où:

- $MC_{éla}(v,k)$ est le nombre d'exemples de l'ensemble d'apprentissage mal classés par le nœud v de T_k dans l'arbre élagué à v.

- MC (v,k) est le nombre d'exemples de l'ensemble d'apprentissage mal classés sous le nœud v dans l'arbre non élagué.
- N(k) est le nombre de feuilles de T_k.
- Nt (v,k) est le nombre de feuilles du sous- arbre de T_k situé sous le nœud v.

Ce critère permet donc d'élaguer un nœud de T_k de façon à ce que $T_{k+1,}$ l'arbre obtenu, possède le meilleur compromis entre taille et taux d'erreur apparent. Finalement, la suite $S=(T_{max},T_1,\ldots,T_k,\ldots\ldots,T_n)$ possède un élément T_k pour lequel le nombre d'erreurs commises est minimal sur l'ensemble de validation: c'est cet arbre-là qui sera finalement retenu par la procédure d'élagage.

Algorithme d'élagage d'un arbre de décision:

Procédure élaguer (T_{max})
k←0
T_k←T_{max}
Tant que T_k a plus d'un nœud faire
Pour chaque nœud v de Tk faire
Calculer le critère *W (T_k,v)* sur l'ensemble d'apprentissage
Fin pour
Choisir le nœud vm pour lequel le critère est minimum
T_{k+1} se déduit de T_k en y remplaçant vm par une feuille
k←k+1
fin tant que
Dans l'ensemble des arbres $\{T_{max},T_1,\ldots..,T_k,\ldots.,T_n\}$, choisir celui qui a la plus petite erreur de classification sur l'ensemble de validation.

Malgré ces qualités, les arbres de décision souffrent des limitations qu'il vaut mieux connaître avant de les utiliser. D'une part, la procédure globale d'apprentissage peut être difficile à appréhender pour les arbres de grande taille (cependant, la classification d'un élément particulier est toujours compréhensible), leur caractère nettement heuristique a donné lieu à un foisonnement de variantes dont les différences, très réelles, sont

difficilement perceptibles au nouveau venu. D'autre part, il n'est pas facile de traiter les données numériques, où le mécanisme de partitionnement repose sur des tests à partir de seuils sur les attributs. Ces seuils rendent la modélisation particulièrement sensible au bruit et à la variabilité des formes en entrée. C'est pourquoi, ils sont plus souvent utilisés dans des domaines où les informations sont représentées par des attributs symboliques. Cependant, l'introduction d'une représentation par sous-ensembles flous a permis d'élaborer des arbres de décision flous particulièrement adaptés et plus efficaces pour traiter les données numériques et symboliques sujettes aux imprécisions.

5. Arbres de décision flous

L'inconvénient principal des arbres de décision tels qu'ils ont été présentés jusqu'ici provient de la façon dont les conditions sur les attributs sont utilisées pour faire les partitionnements.

Ces conditions sont strictes, ne laissant aucune alternative possible pour attribuer un individu à un des fils du nœud considéré. Les partitionnements résultants peuvent ainsi devenir arbitraires, rendant la modélisation moins robuste voir incohérente (et donc difficilement exploitable). Ce phénomène est encore accru si les données manipulées sont imprécises. Plus particulièrement, pour une approche de reconnaissance de forme travaillant dans un espace de représentation numérique, les conditions sur les attributs se traduisent essentiellement par des frontières séparatrices nettes. Celles-ci étant déterminées de façon à optimiser la séparation des classes (par le biais de la mesure de discrimination) elles ne possèdent bien souvent aucun lien explicite avec les propriétés des formes telles qu'elles apparaissent dans l'espace de représentation. Les connaissances qui en résultent manquent donc souvent de robustesse et sont difficiles à interpréter. Les arbres de décision flous apportent alors une solution intéressante à ces différents problèmes (Ragot, 2003).

Afin d'améliorer la robustesse des arbres de décision classiques certains chercheurs ont modifié le mode de représentation de leurs arbres afin d'utiliser des conditions plus souples sur les attributs numériques. Les arbres de décision flous sont alors une extension directe reposant sur le formalisme bien établi de la théorie des sous-ensembles flous. L'idée principale consiste à décrire les attributs par des sous-ensembles flous appelés « *modalités floues* » et à utiliser celles-ci dans les conditions permettant de faire les partitionnements. Par exemple dans la figure 3.1, les deux conditions portant sur le rapport hauteur/largeur du diacritique peuvent être définies par les

deux propositions floues suivantes : “ la forme du diacritique est un point “ (rapport *hauteur_largeur* proche de 1) et “la forme du diacritique est un trait“ (rapport *hauteur/largeur* faible). Au niveau de la structure de l’arbre, les arcs sont alors étiquetés par les sous-ensembles flous (termes linguistiques) correspondants.

Les méthodes liées à l’apprentissage des arbres de décision flous peuvent être considérées comme une extension de celles des arbres de décision classique. On peut distinguer trois évolutions majeures :

- Une *fuzzification* de l’espace de représentation c’est à dire la détermination de sous-ensembles flous pour décrire les attributs qu’ils soient à l’origine symboliques ou numériques et continus.
- L’adaptation de la méthode de partitionnement
- L’adaptation de la mesure de discrimination.

6. Systèmes fondés sur les arbres de décision

Parmi les méthodes d'apprentissage fondées sur les arbres de décision les plus connues, on trouve l'algorithme ID3 proposé par Quinlan en 1979 et la méthode CART (Osório, 1998). Ces systèmes sont assez semblables, ils ont été développés par deux groupes de recherche séparés et presque à la même époque. La principale différence entre ces deux systèmes réside dans le choix de la mesure utilisée pour la sélection des attributs pendant la construction de l'arbre. Cette mesure est généralement fondée sur la théorie de l'information de Shannon (entropie et gain d'information, utilisés dans ID3).

La méthode ID3 est à l'origine de plusieurs autres systèmes. L'un des plus connus est le système C4.5, développé plus récemment par Quinlan (en 1993). A l'origine du système C4.5 se trouve la volonté de résoudre les différents problèmes rencontrés dans son prédécesseur, l'algorithme ID3. Ces problèmes sont assez caractéristiques des arbres de décision en général, à savoir :

- Les premiers systèmes ont été conçus pour traiter des attributs avec des valeurs nominales et discrètes (*variables qualitatives*) et ne pouvaient pas traiter des attributs avec des valeurs continues (*variables quantitatives*). Il manquait des méthodes bien adaptées pour trouver les bonnes questions à poser sur les valeurs continues et qui seront postérieurement associées aux nœuds de l’arbre.

- Le choix du meilleur attribut qui va diviser les exemples présente lui aussi certains problèmes. Comment peut-on nous assurer que la méthode employée va nous amener à la construction de l'arbre le plus simple possible ? Malheureusement, on ne peut pas être sûr de l'obtenir, puisque ce type de problème devient intraitable d'un point de vue calculatoire [OSO 98]. De plus, il n'est pas difficile de trouver des exemples pour lesquels une variable a un très grand pouvoir de discrimination, mais n'aide pas beaucoup à la solution du problème (e.g. l'âge ou le sexe d'une personne permet de discriminer les gens d'une façon très efficace, toutefois est-elle une bonne variable par rapport à tous les types de problèmes de classification?).
- Parfois, on ne connaît pas toutes les valeurs de chacun des attributs considérés pour la classification. Le fait que l'on doit travailler sur des domaines représentés par des informations incomplètes ouvre la problématique du traitement des *attributs avec valeurs manquantes.*
- Les arbres de décision souffrent du problème de sur-apprentissage (*overtrainning/ovefitting*). Le fait que la construction de l'arbre de décision ne s'arrête que lorsque tous les exemples de la feuille appartiennent à une même classe, suppose des ensembles d'apprentissage contenant uniquement des exemples "parfaits" (ils doivent être corrects et doivent bien représenter tout l'espace des entrées**). L'arbre de décision finit par trop se spécialiser dans les exemples d'apprentissage, ce qui peut poser de graves problèmes au niveau de la généralisation aux nouveaux cas.** Souvent, les bases d'apprentissage sont incomplètes et ont des exemples incorrects, et donc, cette méthode d'apprentissage ne fonctionnera pas correctement ;
- Sur un arbre de décision, les connaissances restent "cryptées". Bien qu'il s'agisse d'une représentation symbolique des connaissances, elle peut être parfois un peu difficile à lire et à interpréter.
- Les arbres de décision, tels que ceux construits par ID3, doivent être complètement reconstruits pour prendre en compte un nouvel exemple ajouté à la base d'apprentissage. Ce type d'arbre n'est pas incrémental du point de vue de l'acquisition de données, même si sa structure est construite d'une façon incrémentale (par ajout de nœuds). Il faut recommencer tout l'apprentissage pour prendre en compte un nouvel exemple.
- Malgré le fait que ces méthodes cherchent à construire des arbres simples, certaines branches peuvent être répliquées. La structure en arbre n'est pas toujours la forme la plus simple et économique pour représenter les informations. Les graphes de décision restent une alternative à la structuration en forme d'arbres.

- Les arbres de décision, comme leur nom l'indique, servent à obtenir une classification des données et ont donc une ou plusieurs sorties binaires. On ne peut pas avoir une sortie qui représente une valeur continue. Ce type d'arbres de décision n'est pas utilisable pour l'approximation de fonctions (régression).
- Enfin, ces arbres de décision n'exploitent pas de méthodes permettant l'utilisation des connaissances théoriques disponibles sur le problème. Dans leur forme initiale, les arbres d'induction travaillent uniquement sur les connaissances empiriques.

Plusieurs systèmes fondés sur les arbres de décision ont pris en compte ces différents problèmes décrits ci-dessus, en essayant d'apporter des solutions plus performantes (Osório, 1998) :

- **C4.5**: c'est un système dérivé de l'ID3. Il présente des propositions pour traiter et améliorer les items :
  A. Discrétisation des variables quantitatives.
  B. Prise en compte des coûts associés au choix de chaque attribut.
  C. Prise en compte par la fonction de sélection d'attributs.
  D. Elagage de l'arbre à la fin du processus d'apprentissage, à partir d'un ensemble d'exemples de test de généralisation.
  E. Explicitation de règles symboliques de type Si-Alors à partir des arbres de décision. De plus Ross Quinlan, le concepteur de l'ID3 et C4.5 propose, à travers sa compagne Rulequest, plusieurs versions récentes de C4.5, comme **C5** (sous Unix) et **See5** (sous Windows)[2].
- **Assistant Professional** il utilise une autre fonction de choix d'attributs qui permet l'élagage de l'arbre de décision (*post-pruning*), et implémente une méthode de binarisation des attributs avec des valeurs continues.
- **ID5R** : Il s'agit d'une méthode de construction incrémentale d'arbres de décision qui permet d'apprendre des nouveaux exemples sans avoir besoin de recommencer tout l'apprentissage.
- **ITI**: système d'apprentissage incrémental d'arbres de décision, développé à partir de l'ID5R. Il apporte des améliorations par rapport à l'utilisation de variables continues et de valeurs manquantes, et il permet aussi d'élaguer les arbres.
- **IDL**: il s'agit d'un algorithme de construction incrémentale d'arbres de décision fondé sur la méthode ID5R.

[2] http://www.rulequest.com

- **PRISM**: méthode d'apprentissage fondée sur l'algorithme ID3 de Quinlan. Il se distingue d'ID3 par le type de méthode employée pour la sélection des attributs discriminants.
- **SIPINA**: système de construction de graphes de décision. Il utilise une fonction particulière de sélection d'attributs, implémente une méthode de discrétisation de variables continues ainsi qu'une méthode qui évite le sur-apprentissage et permet aussi l'explicitation de règles représentées dans les graphes de décision.

7. Conclusion

Les arbres de décision fournissent des méthodes effectives qui obtiennent de bons résultats dans la pratique, ils possèdent l'avantage d'être compréhensibles par tout utilisateur (si la taille de l'arbre produit est raisonnable) et d'avoir une traduction immédiate en termes de règles de décision. Pour le système à base de règles induit, les règles sont mutuellement exclusives et l'ordre dans lequel sont examinés les attributs est figé. Les méthodes sont non optimales: les arbres produits ne sont pas les meilleurs. En effet, les choix dans la construction des arbres, basés sur de nombreuses heuristiques, ne sont jamais remis en question (pas de retour en arrière (ou backtraking)). Enfin, il est possible de modifier les valeurs de nombreux paramètres, de choisir entre de nombreuses variantes et faire le bon choix n'est pas toujours aisé.

Enfin, comme la construction d'un arbre de décision permet la génération de systèmes à base de règles, il nous faut faire le lien avec l'approche ***Systèmes Experts***. Les deux approches ***à partir de données*** et **par expertise** peuvent être considérées comme concurrentes et complémentaires:

- L'approche système expert peut être envisagée si l'on dispose d'une expertise suffisante dans le domaine d'application visé et si cette expertise est formalisable en termes de règles. La taille du domaine d'application doit être bien délimitée. L'expérience a prouvé que la maintenance et l'évolution des systèmes experts était une tâche difficile.
- Les méthodes d'apprentissage sont utilisées dans des domaines où les experts n'arrivent pas à dégager les règles qu'ils utilisent. Les règles sont générées à partir de données (souvent des données historiques pour le problème).
- Il arrive également que ces deux approches soient utilisées conjointement: des experts seront souvent de bon conseil pour dégager les attributs pertinents relativement à un problème donné. Ainsi, dans certains cas, des systèmes d'apprentissage produiront automatiquement des règles qui pourront être directement insérées dans un système expert.

Précédemment, nous avons utilisé la technique des arbres de décision pour la reconnaissance de l'écriture arabe manuscrite (Amrouch et al., 2011). Et dans ce travail de thèse, nous allons utiliser les arbres de décision dans le contexte de la fusion d'ontologies pour détecter les classes similaires candidates à la fusion. La justification du choix de cette technique ainsi que la méthode de son application seront détaillées dans les chapitres suivants.

Chapitre 4 . Conception du Framework AOM-FOM

1. Introduction

Pour s'adapter à la nature dynamique du web sémantique et de ses applications, l'automatisation complète du processus de la fusion d'ontologies est une exigence. La contribution présentée dans ce chapitre évite l'intervention humaine en automatisant complètement le processus d'identification des mappings entre entités similaires ainsi que la construction de l'ontologie globale issue de la fusion.

Nous proposons une nouvelle combinaison de stratégies différentes pour détecter les entités similaires en utilisant toutes les sémantiques modélisées par les bases de connaissances ontologiques, notamment les classes, les propriétés, les instances ainsi que leurs structures de hiérarchisation. Selon les chercheurs, il n'existe pas une meilleure solution pour le problème de la fusion d'ontologies (Raunich and Rahm, 2012). Mais tout dépend des exigences de l'application qui la nécessite. Cependant, dans tout processus de fusion d'ontologies, le défi majeur est lié à la détection de classes similaires qui vont être fusionnées en une seule classe.

Dans le monde réel, divers problèmes peuvent affecter la qualité des résultats de mappings parmi lesquels :

a) Les noms des entités ontologiques sont souvent formés par des mots composés tels que *ProgramCommitteeMember*, *SessionChair*, *isWrittenBy*, *etc.*
b) Les noms des entités sont codés par des abréviations ou par les initiales de noms tels que PC_Member pour *ProgramCommittee_Member*.
c) Des entités différentes peuvent être étiquetées par le même nom tel que *PostGraduate* (pour décrire un étudiant qui continue ses études après la graduation, *PostGraduate Student)* et *PostGraduate* (pour décrire un cours dédié aux étudiants en deuxième cycle, *PostGraduate courses*), ce qui entraîne des mismatches (incompatibilités) de modélisation.

d) Des entités similaires peuvent être classifiées différemment dans leurs hiérarchies comme par exemple *Phd-Researcher* qui est classifié comme étant (*is-a student*) dans une ontologie et comme étant (*is-a employee*) dans la deuxième ontologie, ce qui entraîne des mismatches (incompatibilités) de conception.

En plus de ces obstacles, les systèmes actuels de fusion d'ontologies intègrent les techniques basées sur l'alignement des instances pour améliorer les résultats de mappings. Ils utilisent l'analyse linguistique basée sur le traitement du langage naturel (TLN) pour détecter les similarités entre toutes les paires d'instances. Cependant, vu que les techniques basées sur le TLN sont dépendantes d'un contexte spécifique, ceci peut limiter les sémantiques des mappings obtenus à ce contexte.

A partir de ce qui précède, nous remarquons que l'alignement des noms échoue souvent à prédire des similarités réelles et peut également produire des similarités incorrectes, ce qui aboutit à une ontologie fusionnée erronée.

Pour surmonter ces problèmes, nous proposons une nouvelle approche pour la fusion automatique des ontologies OWL. Pour détecter les classes similaires qui doivent être fusionnées, notre système est basé sur la combinaison de plusieurs modules : Un module de détection de similarités au niveau entité (lui-même est basé sur la combinaison du module de détection de similarités au niveau schéma et le module de détection de similarités au niveau extensionnel) et le module de détection de similarités au niveau structurel. Les similarités ainsi obtenues sont d'abord validées avant d'être utilisées par le module de la fusion.

Dans ce chapitre, nous allons présenter en détail notre contribution pour la fusion automatique d'ontologies OWL-DL. Après avoir introduit ce chapitre, nous allons décrire en détails les caractéristiques de notre système de fusion dans la prochaine section en présentant en particulier nos choix de solutions pour les problèmes de fusion évoqués précédemment et en justifiant ces choix. Nous y présentons aussi les grandes étapes de fusion envisagées. Suite à cela, nous décrirons l'architecture générale de notre système de fusion automatique tout en détaillant ses différents modules.

2. Le système de fusion d'ontologies AOM-FOM

Notre système de fusion automatique d'ontologies est composé de trois étapes principales : Après avoir importé les ontologies sources sous format OWL-DL, elles sont tout d'abord prétraitées pour préparer l'étape de découverte de similarité. Le défi majeur de cette étape est de confronter le premier type de problèmes cité en introduction (problème des noms composés) à travers les techniques de lemmatisation, élimination et normalisation. Ensuite, les mappings du niveau entité (la liste initiale des mappings) sont obtenus en combinant les mappings du niveau schéma (qui détectent les similarités entre les classes et les propriétés en utilisant l'analyse lexicale et/ou sémantique de leurs noms et/ou descriptions) et les mappings du niveau extensionnel (qui détectent les similarités entre les classes en se basant sur leurs instances). En effet, pour résoudre le deuxième type de problèmes cité en introduction (impossibilité de détection des mots dans les dictionnaires et/ou les BDDs lexicales), la technique qui est basée sur l'alignement des instances est basée sur l'apprentissage automatique des instances en utilisant les arbres de décision, pour classifier les instances de l'ontologie cible avec leurs instances similaires (des classes similaires) de l'ontologie source. La classification des instances basée sur les arbres de décision évite les lacunes des techniques de TLN exploitées par les méthodes linguistiques, car elle est indépendante de toutes bases de données lexicales et/ou de dictionnaires de synonymie. En plus, l'exécution parallèle de ces algorithmes (inter et intra niveau) va surement réduire la complexité du temps d'exécution et optimise les performances du processus d'alignement. Le choix du classifieur basé sur les arbres de décision est basé sur le fait qu'il n'est pas une boite noire (tel que les réseaux de neurones), et la classification peut être comprise en parcourant l'arbre de la racine jusqu'aux feuilles. En plus, l'arbre de décision obtenu est traductible sous forme de règle (si…alors) ce qui rend l'arbre plus lisible et compréhensible. En outre, ce classifieur ne demande pour son exécution aucun attribut ou valeur numérique (tel que le nombre de couches, le nombre de neurones, la fonction d'activation, etc.), ce qui n'est pas le cas avec d'autres classifieurs tel que les SVM, les logiques de régression (logistic regression)

et les réseaux de neurones. En plus, les arbres de décision sont le seul modèle permettant de gérer de manière homogène les attributs de natures variées, en particulier les mélanges continus et binaires (Cornuéjols, Miclet, & Kodratoff, 2010).

Dans le même contexte de résolution du deuxième type de problèmes cité en introduction, et si les instances de classes ne sont pas disponibles (pour des raisons de sécurité, par exemple), les mappings obtenus au niveau entité sont enrichies par les mappings obtenus au niveau structurel, en propageant (en diffusant) les similarités précédemment obtenues à travers les structures des ontologies sources (pour détecter plus de mappings entre classes et propriétés) en appliquant quelques heuristiques spécifiques.

Nous avons appliqué l'aspect lexical (ou syntaxique) et sémantique pour la fusion automatique et semi-automatique des ontologies, dans des travaux précédents, à savoir (Amrouch et al., 2012c), (Amrouch et al., 2012d), (Amrouch et al., 2012e) et (Amrouch et al., 2013). Mais l'intégration des aspects extensionnel et structurel est une initiative et une nouveauté de ce travail de thèse (Amrouch et al., 2016).

Pour résoudre le problème de détection de mappings erronés (problèmes 3 et 4, cités en introduction), les mappings obtenus sont sémantiquement vérifiés et raffinés (en appliquant certains patterns de détection des mappings erronés) et ensuite combinés avant d'être exploités par le dernier module de fusion d'ontologies, où les classes similaires seront fusionnées et les classes différentes seront directement copiées dans l'ontologie résultat de la fusion.

Parmi nos autres objectifs est de bien gérer le temps d'exécution ainsi que l'espace mémoire utilisé par notre système. En effet, pour détecter les classes similaires, nous suivons la technique de la recherche d'information où chaque classe de la première ontologie est comparée avec toutes les classes de la deuxième ontologie, ce qui termine par faire $n_1 * n_2$ comparaisons en tout (tel que n_1 est le nombre de classes de la première ontologie et n_2 est le nombre de classes de la deuxième ontologie). Alors, pour minimiser le temps d'exécution écoulé ainsi que l'espace mémoire consommé, nous donnons une priorité de parcourir les classes dans les partitions disjointes.

Vu que notre algorithme de fusion automatique d'ontologies est basé sur un algorithme flexible de mapping, nous l'avons nommé AOM-FOM pour (Automatic Ontology Merging based on a Flexible Ontology Matching).

Dans cette section, nous allons présenter et décrire notre système de fusion automatique d'ontologies AOM-FOM. Ensuite, nous allons détailler les différents modules qui composent son processus tout en décrivant ses différents algorithmes et techniques. Nous terminons par une comparaison conceptuelle de notre système avec les différents systèmes de fusion d'ontologies existant dans la littérature.

2.1. Description du système de fusion d'ontologies AOM-FOM

Comme son nom l'indique, notre système AOM-FOM est un Framework de fusion automatique d'ontologies basé sur un algorithme flexible d'alignement d'ontologies. Son architecture globale est illustrée par la figure 4.1.

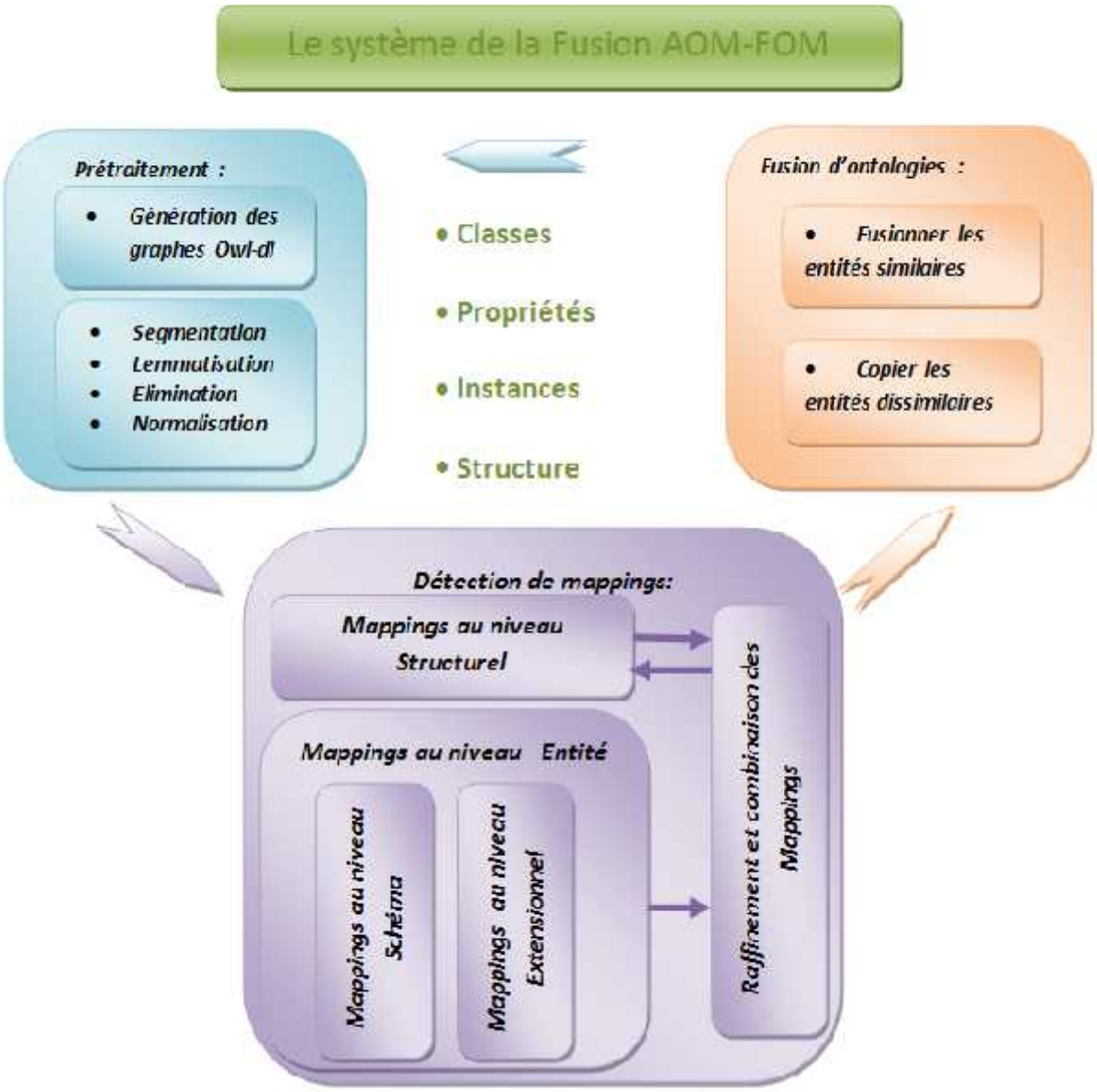


Figure 4-1. L'architecture globale du système proposé, AOM-FOM

Comme présenté par la figure 4.1, notre système est structuré en trois modules correspondant à trois phases principales : le prétraitement, la détection de mappings et la fusion.

- Lors de la première phase, les ontologies sources sont prétraitées : Les graphes OWL-DL sont générés en utilisant l'API *jena* pour faciliter l'exploration et la manipulation des entités ontologiques par les différents algorithmes du système. En plus, les chaînes de caractères étiquetant les entités ontologiques (classes et propriétés)

sont passées à un processus de prétraitement en utilisant l'API *MorphAdorner* (version 1.0) pour préparer la phase de détection de similarités. Il a pour utilité principale de ramener les noms des entités ontologiques à une forme de base (racine), ce qui permet de détecter plus de mappings et alors donner des résultats plus précis. Le processus de *lemmatisation* supporté par l'API *MorphAdorner* est utilisé pour détecter les formes de base des noms et des verbes irréguliers étiquetant les entités ontologiques. Par exemple, la lemmatisation du mot « *sports* » donne la forme de base « *sport* », et la lemmatisation des propriétés (*reviewed*, *reviewing*, *reviews*, *reviewer* et *review*) donne la forme de base « *review* ». Les résultats de prétraitement sont efficacement utilisés par le module de détection de mappings.

- Le module de détection de mappings est basé sur la combinaison de plusieurs modules individuels de détection de similarités. Certains sont basés sur l'analyse lexicale et/ou sémantique des descriptions et des noms de classes et de leurs différentes propriétés. Un autre type d'algorithme de détection de similarité est basé sur l'apprentissage automatique des instances de classes. Le troisième type des algorithmes de détection de similarité est basé sur l'analyse structurelle des ontologies candidates à la fusion. L'analyse lexicale applique les différentes mesures de distance entre chaînes de caractères (telle que la *distance d'édition*, *levenstein*, *jaro*, *jaro winkler*, *etc*.). Alors que l'analyse sémantique est basée sur l'extraction des synonymes des mots à partir d'une base de données lexicale, qui est le dictionnaire *wordnet*[3] (version 3.0). L'accès et l'extraction des synonymes à partir de *wordnet* sont supportés par l'API *JWNL*[4] (version 1.4.1). L'apprentissage automatique des instances de classes est basé sur la technique des arbres de décision. Le module basé sur l'apprentissage automatique des instances de classes par les arbres de décision est indépendant de toutes bases de données lexicales (dictionnaires de synonymes) et/ou de mesure de calcul de distance entre chaîne de caractères, ce qui permet d'éviter les limites des techniques de *TLN* appliquées par les méthodes basées sur l'analyse linguistique

[3] http://wordnet.princeton.edu
[4] http://sourceforge.net/projects/jwordnet/

(lexicale et sémantique). L'analyse structurelle applique certaines heuristiques pour propager les similarités obtenues par les premiers algorithmes à travers les structures des ontologies sources pour détecter plus de mappings. A l'issue de cette étape, un ensemble de patterns de vérification et de détection de mappings erronés est appliqué pour éliminer les mappings incorrects, le plus possible.

- Enfin, les classes qui se correspondent (c-à-d les classes similaires) dans la liste raffinée et combinée de mappings sont fusionnées en une seule classe et les classes n'ayant aucune classe similaire (dans l'autre ontologie) sont directement copiées dans l'ontologie résultat de la fusion. A ce stade, l'ontologie globale issue de la fusion est validée en utilisant un moteur de raisonnement open source, basé sur la logique de description (DL), Racer++, pour s'assurer que l'ontologie résultat est correcte et consistante.

A l'instar de PROMPT, notre système peut fonctionner dans un mode semi-automatique. Dans ce cas, le système génère automatiquement la liste initiale de mappings et demande le feedback de l'expert humain.
A l'instar de plusieurs systèmes de fusion d'ontologies (tel que PROMPT), notre système suit une approche cyclique pour fusionner les ontologies sources.

2.2. Le processus de fusion d'ontologies AOM-FOM

Comme tout processus de génie logiciel, notre processus de fusion automatique d'ontologies, AOM-FOM est modélisé à travers un modèle de conception cyclique, composé de plusieurs étapes successives, où le résultat de chaque étape est exploité par l'étape suivante. Ces différentes étapes sont schématisées par la figure 4.2 et détaillées ci-dessous :

Figure 4-2. Le processus de fusion d'ontologies AOM-FOM

2.2.1. Analyse et conception

Lors de cette étape, les ontologies sources sont importées dans notre Framework sous format OWL-DL. Ensuite, leur contenu est analysé et modélisé et les noms des entités ontologiques sont prétraités.

2.2.1.1. Élaboration du modèle conceptuel

Une des actions qui facilite l'alignement ainsi que la fusion des ontologies sources est leur modélisation à travers un modèle conceptuel plus facile à lire et à traiter par les différents algorithmes du système. Le modèle conceptuel le plus adéquat est ce qu'on appelle « Le graphe OWL-DL ». Ce dernier est automatiquement généré en utilisant le Framework du web sémantique, jena (version 1.0) et/ou l'API owl-API (version 2.0). Alors que PROMPT utilise le modèle RDFS, notre Framework utilise les graphes OWL-DL .

2.2.1.2. Prétraitement des termes ontologiques

Pour faciliter le processus de comparaison des termes ontologiques étiquetant les classes ainsi que leurs propriétés (i.e. calculer les distances entre leurs chaînes de caractères), il est très important d'effectuer un certain nombre d'opérations de prétraitement. Elles améliorent significativement les résultats d'alignement. En plus, lors de l'alignement basé sur l'extraction des synonymes, les opérations de prétraitement facilitent leur reconnaissance par les bases de données lexicales et/ou dictionnaires de synonymes. Les différentes opérations de prétraitement effectuées par notre système sont :

1. **La normalisation** : Il s'agit de transformer tous les caractères des termes ontologiques en minuscule. Par exemple, la normalisation du terme « *ETUDIANT* » ou « *Etudiant* » donne la forme « *étudiant* ».
2. **Élimination (stripping) des caractères spéciaux, espaces, chiffres** *:* comme indiqué par le titre, il s'agit de supprimer les caractères spéciaux contenus dans les termes ontologiques, tels que les *apostrophes*, les *traits*, les *tirets*, etc., par exemple, « *MS-STUDENT* » sera « *MSSTUDENT* » et {« *MS* », « *STUDENT* »}. Les chiffres, les espaces, les tabulations, etc., sont aussi éliminés. Par exemple, « MacDonald4U » sera {« *MacDonaldU* »} et {« *MacDonald* », « *U* »}.
3. **Élimination du « *And* »** : « *And* » ou « *&* » est éliminé des termes ontologiques. Par exemple « *Photo and Camera* » ou « *Photo & Camera* » retourne un ensemble de deux éléments {Photo, Camera}.
4. **Lemmatisation (Tokenization) :** Pour qu'ils soient reconnus par les dictionnaires des synonymes ou les bases de données lexicales, lors de l'alignement basé sur les synonymes, les mots composés ont besoin du processus de lemmatisation (extraction des racines de leurs sous mots). Par exemple, si les termes ontologiques « *CsDepartment* » ou « *PhdStudent* » sont recherchés dans *wordNet* , aucun résultat ne sera obtenu. La lemmatisation de ces mots vers {*Cs, Department*} et {*Phd, Student*} sera utile et permet d'attribuer des sens (sémantiques) propres à ces termes.

5. **Élimination des mots d'aide**: Ce processus est généralement utilisé pour prétraiter les noms de propriétés. Généralement, les noms de propriétés utilisent les prépositions, les articles ou les mots d'aide pour ajouter ou montrer des expressions sémantiques entre les entités. Par exemple, *RegistersIn* (*Student*, *Course*), *ReviewedBy* (*Paper*, *Reviewer*), *hasAddress* (*Person*, *Location*), etc. Il est alors très utile d'éliminer les mots d'aide tels que « *In* », « *By* », « *From* », « *has* », etc.

Les opérations de prétraitement (tels que la normalisation, l'élimination ou le stripping et la lemmatisation) qui donnent un ensemble de labels, demandent une attention spéciale lors du mapping. En effet, tous les labels d'un ensemble doivent être alignés avec tous les labels du deuxième ensemble. Par exemple, le nom de classe « *Photo and Cameras* » ou « *Camera & Photos* » retournent deux éléments dans chaque ensemble : {Photo, *Cameras*} et {*Camera*, *Photos*}, ce qui exige que tous les éléments du premier ensemble soient alignés avec tous les éléments de deuxième ensemble : (*Cameras=Camera*) et (*Photo=Photos*).

2.2.2. Découverte de mappings

Il s'agit de l'étape la plus importante dans le processus de fusion d'ontologies. Elle vise à identifier les classes candidates pour la fusion en se basant sur les mappings entre les ontologies sources. Dans notre processus, la découverte de mappings passe par les étapes suivantes :

2.2.2.1. Analyse de l'espace de recherche de similarité

La plupart des méthodologies des systèmes de fusion d'ontologies, tel que PROMPT, appliquent une analyse exhaustive pour la détection des classes similaires entre ontologies sources, où chaque classe de la première ontologie est comparée avec toutes les classes de la deuxième ontologie. En d'autres termes, pour deux ontologies sources o_1 de n_1 classes et o_2 de n_2 classes, on effectue n_1*n_2 comparaisons pour détecter les mappings candidats à la fusion. Cependant, cette stratégie augmente exponentiellement le temps d'exécution ainsi que l'espace mémoire utilisé pour la détection des mappings

candidats à la fusion. Alors, pour minimiser la complexité en temps d'exécution et en espace mémoire de stockage, nous avons opté pour une stratégie basée sur l'analyse des partitions ontologiques disjointes. Par exemple, considérons l'ontologie « *personnel* » de la figure 4.3 où l'axiome de disjonction partitionne le personnel d'un département en deux partitions disjointes *étudiant* et *professeur*.

Figure 4-3. Les partitions disjointes dans les ontologies Personnel

Après que notre algorithme détecte les mappings (*Etudiant*, *Elève*) et (*Professeur*, *Enseignant*), et en nous basant sur l'axiome de disjonction entre *étudiant* et *professeur* (resp. entre *élève* et *enseignant*), il cherche les mappings des sous classes (*étudiant*) sous la hiérarchie de la classe (élève) et non pas sous la hiérarchie de la partition disjointe (*enseignant*) ni dans l'ontologie complète. Cette stratégie s'avère très performante surtout dans le cas de grandes ontologies qui contiennent des milliers de classes. Plus il y-a des axiomes de disjonction dans les ontologies sources, plus l'espace de recherche sera plus limité, ce qui réduit efficacement le nombre de comparaisons pour détecter les mappings. Cette stratégie fonctionne efficacement lorsque les ontologies sources ne contiennent pas des hétérogénéités de conceptualisation du domaine d'application. Dans le cas contraire, on risque de ne pas détecter certains mappings corrects, tel qu'illustré, par l'exemple de la figure 4.4, où dans la deuxième ontologie, la classe (*doctorant*) est classifiée directement sous la classe (*personnel*) (ou même sous la classe (*enseignant*) si un *doctorant* peut enseigner)

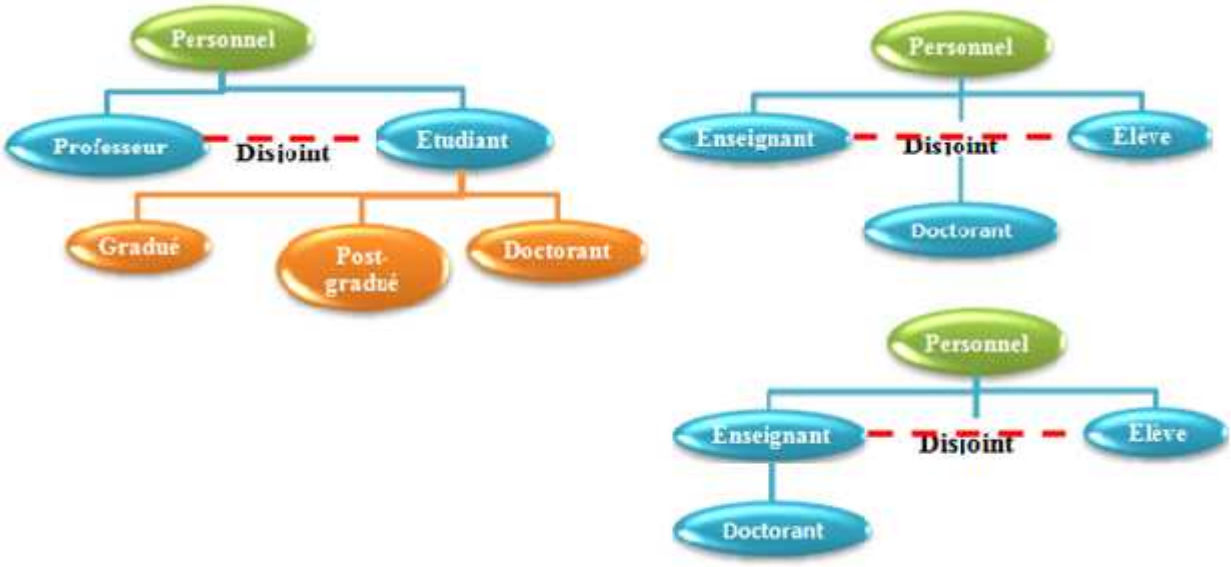


Figure 4-4. Hétérogénéité conceptuelle lors de la classification de la classe 'Doctorant' et partitions disjointes

Dans ce cas là, pour éviter de rater des mappings, on cherche dans les classes du niveau supérieur et ainsi de suite jusqu'au balayage de toute l'ontologie.

Généralement, la stratégie adoptée par notre système est basée sur un mécanisme de priorité qui cherche les mappings dans les partitions disjointes. Si les mappings ne sont pas détectés, on cherche alors dans les siblings (partitions de même niveau) ou dans les niveaux supérieurs. (Cette approche ressemble à la recherche exhaustive dans tout l'espace mais effectuée pas à pas).

2.2.2.2. Calcul de similarité et agrégation de similarité

2.2.2.2.1. Calcul de similarité: Le module de découverte de mappings est principalement basé sur l'étape de calcul de similarité entre les classes ainsi que entre leurs propriétés (propriétés d'objet et propriétés de type de données) dans les ontologies sources.

Cependant, vue la variation de la structure et du contenu ontologique, utiliser un seul type de mesure de similarité est insuffisant et ne donne pas de très bonnes performances. Alors, contrairement à PROMPT qui utilise uniquement une mesure de similarité lexicale entre chaînes de caractères et détecte seulement les chaînes de caractères identiques, notre

système est basé sur une stratégie hybride pour le calcul de similarité entre les classes et leurs propriétés (DTP et OP). Il combine les résultats de similarités au niveau structurel avec les résultats de similarités au niveau entité. La similarité au niveau entité, elle-même combine les résultats de similarités au niveau schéma, calculées sur les classes et leurs propriétés avec les similarités au niveau extensionnel, calculées sur les instances de classes. La similarité calculée sur les instances est basée sur l'apprentissage automatique alors que toutes les autres similarités sont basées sur l'analyse lexicale et /ou sémantique des chaînes de caractères :

Les techniques basées sur l'analyse lexicale et/ou sémantique des chaînes de caractères peuvent être confrontées à différentes limites des techniques de traitement automatique de langue naturelle. C'est pourquoi nous les avons combinées avec une nouvelle technique basée sur l'apprentissage automatique des instances en utilisant les arbres de décision qui est indépendante de tout type de traitement linguistique (lexicale ou sémantique) pour supporter les techniques linguistiques.

Dans ce qui suit nous expliquons l'application des différents mesures et techniques de calcul de similarité appliquées dans notre système :

1) Calcul de similarité au niveau entité : Cette étape détecte les similarités entre les classes, leurs propriétés ainsi que leurs instances dans les ontologies sources. Elle combine les résultats de similarités calculées au niveau schéma (calculées sur les classes et les propriétés) avec les résultats de similarités calculées au niveau extensionnel (calculées sur les instances) :

a) Calcul de similarité au niveau schéma : Elle détecte les similarités entre les classes et les propriétés dans les ontologies sources :

- **La similarité entre les classes***:* En OWL-DL, une classe est décrite par différents nœuds. Principalement on trouve son nom (LocalName), son Identificateur (ID) et sa description (comment). Le nom de la classe est hautement représentatif et significatif. Il contribue par le poids le plus important pour sa description. Cependant, il peut être codé par des abréviations ou même des chaînes de caractères aléatoires qui ne peuvent pas être reconnus par les dictionnaires et/ou les bases de données lexicales utilisées. Alors, pour

calculer la similarité entre deux classes on considère le nœud qui rend la valeur de similarité maximale :

$$S(c_1, c_2) = Max(S_{name}, S_{comment}, S_{ID}) \quad (1)$$

- **La similarité entre les propriétés (S_{dtp} et S_{op}):** Les propriétés de type de données (data type properties, dtp) et les propriétés d'objets (object properties, op) caractérisant une classe dans une ontologie représentent sa sémantique et son contexte. Les propriétés de type de données d'une classe dans une ontologie jouent le même rôle des attributs d'une classe dans un modèle Entité/Association. Par exemple, chaque classe (étudiant) est décrite par les propriétés de type de données (attributs) suivantes : code, nom, âge, adresse, etc. Alors que les propriétés d'objets d'une classe dans une ontologie sont similaires aux associations (ou relations) dans un modèle Entité/Association. Elles mettent en relation directes ou réciproques les classes d'ontologies. Par exemple, les propriétés d'objet de la figure 4.5 *contribuer (auteur, article)* et *Est-Évalué-Par (article, examinateur)* mettent en association (*auteur* et *article) e*t *(article* et *examinateur)*, respectivement et représentent leurs descriptions réelles. Ces propriétés sont assez performantes pour aider l'algorithme de fusion à détecter des mappings corrects entre ontologies sources.

Dans une ontologie OWL-DL, une propriété est mieux schématisée sous forme d'un graphe composé des nœuds suivants (i) le domaine de la propriété, (ii) le nom de la propriété et (iii) le Co-domaine (the range) de la propriété.

Exemple 1 : la classe « *étudiant* » a la propriété de type de donnée *« code »* de type *« string »*.

Dans cet exemple, le domaine de la propriété de type de donnée « *code* » est « *étudiant* » et son co-domaine est « *string* ».

Exemple 2 : La classe *« Audience »* est reliée avec la classe *« Auteur »* par la propriété d'objet *« Donner un feedback »* (voir figure 4.5).

Dans cet exemple, le domaine de la propriété d'objet *« Donner un feedback »* est *« Audience »* et son co-domaine est *« Auteur »*

Figure 4-5. Association de différentes classes par les propriétés d'objet

Alors, la similarité entre deux propriétés est calculée en prenant le maximum des similarités entre ces nœuds. En d'autres termes, la similarité entre deux propriétés de type de données est égale à la valeur maximale des similarités entre : leurs noms (S_{name}), les classes de leur domaine (S_{dmne}) et de leurs co-domaine (range) (S_{rge}).

Notons que ces similarités sont calculées en se basant, toujours, sur l'analyse lexicale et /ou sémantique des chaînes de caractères étiquetant leurs noms. La similarité entre deux propriétés de type de données est alors calculée selon l'équation suivante :

$$S_{PTD}(p_1, p_2) = Max(S_{name}, S_{dmne}, S_{rge}) \qquad (2)$$

Le calcul de similarité au niveau schéma effectue alors, une analyse linguistique (lexicale et sémantique) des chaînes de caractères étiquetant les nœuds des classes et les nœuds de leurs propriétés dans les ontologies sources. La similarité obtenue sur chaque nœud est le maximum entre la similarité lexicale et la similarité sémantique :

- **Similarité lexicale :** En utilisant l'API *SimMetrics*, nous avons utilisé une grande variété d'algorithmes pour calculer la similarité lexicale entre les chaînes de caractères étiquetant les nœuds des classes (*Id*, *name*, *comment*) et de leurs propriétés (*name*, *domain*, *range*) dans les ontologies sources. La plupart de ces mesures sont basées sur la même supposition décrite par (Maedche et Staab, 2002) qui affirme que deux chaînes de caractères sont similaires si elles partagent suffisamment d'éléments importants.

- **Similarité sémantique:** En utilisant l'API *JWNL*, nous avons exploité la BDD, wordnet, pour calculer la similarité sémantique entre les chaînes de caractères étiquetant les nœuds des classes et de leurs propriétés dans les ontologies sources.

Une fois que les similarités lexicales et sémantiques sont calculées, leur agrégation est effectuée en appliquant l'équation (3) :

$$SIM_{lexsem}(c_1, c_2) = Max(Sim_{sem}(c_1, c_2), Sim_{lex}(c_1, c_2)) \qquad (3)$$

b) Calcul de similarité au niveau extensionnel: Elle enrichit les résultats de similarité obtenus au niveau schéma en détectant plus de mappings entre classes similaires en se basant sur leurs instances. Pour mesurer la similarité basée sur les instances, nous avons intégré dans notre système un module d'apprentissage automatique des instances en utilisant les arbres de décision pour classifier les classes (à travers la classification de leurs instances) d'une ontologie avec leurs classes similaires de l'autre ontologie. Pour ce faire, chaque classe (des deux ontologies sources) est représentée par le vecteur de ses instances. Le module d'alignement au niveau extensionnel est illustré par la figure 4.6.

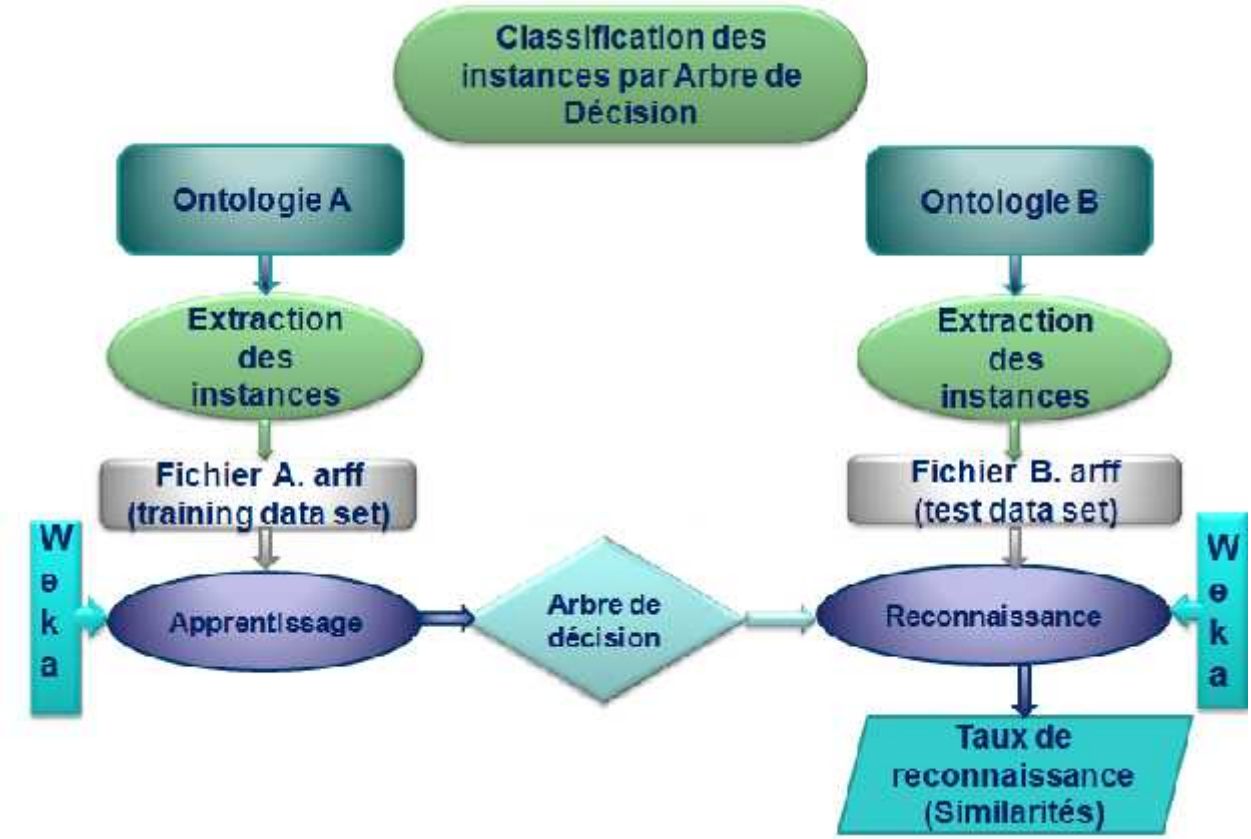


Figure 4-6. Le module d'alignement au niveau extensionnel

Comme illustré par la figure 4.6, les instances de la première ontologie, attribuées à leurs classes respectives sont organisées dans un fichier .ARFF. Ce dernier est alors utilisé comme étant un ensemble d'apprentissage pour construire le classifieur basé sur les arbres de décision lors de l'étape d'apprentissage. Alors que les instances des classes de la deuxième ontologie sont codées dans un autre fichier .ARFF. Ce dernier est utilisé comme étant un ensemble de reconnaissance (et/ou de test) pour classifier, éventuellement, une classe de la deuxième ontologie (à travers la classification de ses instances) avec sa classe similaire de la première ontologie. A ce stade, la similarité basée sur les instances (S_{inst}) entre une classe c_2 de l'ontologie o_2 et une classe c_1 de l'ontologie o_1 est égale au taux d'instances de la classe c_2 de l'ontologie o_2 qui sont reconnues ou classifiées comme étant des instances de la classe c_1 de l'ontologie o_1. La classe c_1 de l'ontologie o_1 qui est similaire à la classe c_2 de l'ontologie o_2 est celle qui correspond au

taux le plus élevé des instances classifiées des classes c_2 de l'ontologie o_2. Ce taux doit excéder un seuil critique.

Nous rappelons que l'algorithme d'apprentissage le plus connu et le plus utilisé pour construire un classifieur basé sur les arbres de décision utilise l'algorithme de Quinlan c4.5 (chap. 3§ 3) que nous avons implémenté par weka, j48 (chap 5 § 1.6).
Cet algorithme est divisé en trois étapes qui dépendent de l'algorithme choisi :

i. Croissance de l'arbre (séparation en arcs basée sur la sélection de l'attribut le plus discriminant pour séparer les ensembles et sous ensembles d'apprentissage).
ii. Arrêt de la croissance
iii. Elagage de l'arbre.

Le critère de séparation des arcs de l'arbre (choix de l'attribut le plus discriminant) est la mesure la plus importante pour catégoriser les différents algorithmes de construction des arbres de décision.

2) **Calcul de similarité au niveau structurel :** Pour détecter plus de mappings entre les classes et les propriétés similaires en se basant sur les similarités au niveau structurel, nous nous sommes inspirés de l'intuition que « Les éléments de deux modèles sont similaires quand leurs éléments adjacents sont similaires ». En d'autres termes, une partie de la similarité entre deux éléments est propagée à leurs voisins, respectifs (Sergey, Hector and Erhard, 2002). En effet, le calcul de similarités au niveau structurel commence une fois que les similarités au niveau entité sont calculées et vérifiées. Le module de calcul de similarité au niveau structurel diffuse alors, les similarités précédemment calculées à travers les structures des éléments sources, en appliquant les heuristiques suivantes dans cet ordre et d'une manière récursive jusqu'à ce qu'aucun autre mapping ne soit détecté :

1) Si la chaîne de caractères étiquetant une classe source est incluse dans la chaîne de caractères étiquetant une classe cible, et les deux classes ont au moins deux parents qui sont similaires (s'ils existent sinon un seul) alors les deux classes sont similaires.

2) Si deux classes (source et cible) partagent les mêmes propriétés (DTP et OP) alors elles sont similaires.

3) Deux propriétés (DTP et OP) sont similaires si leurs domaines sont similaires et leurs co-domaines sont également similaires. Nous excluons de cette heuristique les propriétés de type de données qui ont le même co-domaine (type) dans le même domaine (la classe qu'ils décrivent), pour éviter de détecter plus de mappings incorrects.

4) Si deux propriétés d'objet sont détectées comme étant similaires et leurs domaines sont similaires alors leurs co-domaines sont aussi similaires.

5) Si deux propriétés d'objet sont détectées comme étant similaires et leurs co-domaines sont similaires alors leurs domaines sont aussi similaires.

2.2.2.2.2. Agrégation de similarité

Une fois que toutes les similarités entre les classes et leurs propriétés dans les ontologies sources sont calculées, leur agrégation est une étape inévitable afin qu'on puisse trouver la combinaison des résultats de similarités entre les classes qui correspondent l'une à l'autre.

Contrairement à d'autres systèmes tels que PROMPT, qui n'utilisent qu'une seule mesure de similarité, ne demandant ainsi aucune opération d'agrégation. Notre système est basé sur une multitude de techniques et de mesures de similarité, lexicale, sémantique, apprentissage automatique et similarité structurelle, appliquées sur les différentes entités ontologiques (classes, propriétés de type de données, propriétés d'objets, instances de classes ainsi que les structures des ontologies sources). Il résulte alors plusieurs valeurs de similarités différentes. C'est à ce stade où apparait la nécessité d'agrégation pour trouver la similarité totale entre classes et propriétés. Nous avons décrit et présenté dans chapitre2 §. 4.3, trois méthodes d'agrégation qui peuvent être appliquées pour combiner les mesures de similarité : l'agrégation simple (combinaison linéaire), la moyenne et la moyenne pondérée (combinaison parallèle et hybride).

AOM-FOM est basé sur un algorithme flexible de mapping. Après le choix et l'application des différentes mesures de similarité, nous avons suivi une méthode simple

pour l'agrégation des différentes valeurs de similarité calculées en prenant la valeur de similarité maximale parmi les valeurs de similarité individuelles produites par toutes les mesures choisies et appliquées. Par exemple, si la mesure de similarité lexicale des noms de classes produit une valeur égale à 1.0, alors que la valeur de similarité basée sur la classification des instances est égale à 0.6 alors le système retient la valeur de similarité maximale (i.e. 1.0) et affiche la valeur agrégée à l'utilisateur.

2.2.2.3. Interprétation :

Après que toutes les mesures de similarités entre les paires de classes et propriétés candidates sont calculées et combinées, elles ne peuvent pas être exploitées par le module de la fusion d'ontologies avant qu'elles ne soient interprétées. A l'instar de PROMPT, la phase d'interprétation de notre système est basée sur un critère principal qui est la définition de la valeur d'un seuil critique qui ne peut être identifiée qu'expérimentalement. Les paires de classes et les paires de propriétés ayant une valeur de similarité supérieure ou égale à la valeur du seuil critique sont considérées comme étant des paires candidates à la fusion. Elles sont alors présentées à l'utilisateur sous forme d'une liste initiale de mappings. Mais, elles ne peuvent être présentées au processus de fusion qu'après leur validation. Alors, la phase d'interprétation est principalement basée sur l'étape de validation. En d'autres termes, même si deux classes sont fortement similaires, elles ne peuvent être interprétées comme étant similaires qu'après leur validation et donc ce sont uniquement les mappings valides qui sont présentés à la fusion automatique.

2.2.3. Validation des mappings

La validation des mappings est une étape très importante pour extraire la liste finale à partir de la liste initiale de mappings. C'est à base de cette liste finale qui contient les mappings les plus exacts et les plus consistants qu'on peut obtenir une ontologie fusionnée cohérente, consistante et complète. Alors, pour détecter les correspondances

erronées et raffiner les mappings obtenus, nous avons appliqué trois patterns de vérification sémantique qui ont été introduits dans (Ngo, 2012):

a) Deux mappings (a_1, b_1) et (a_2, b_2) produisent un conflit de type crisscross si a_1 est un père de a_2 dans l'ontologie O_1 et b_2 est un père de b_1 dans l'ontologie O_2.
b) Deux mappings (a_1, b_1) et (a_2, b_2) produisent un conflit de type disjonction/subsumption si a_1 est un père a_2 dans l'ontologie O_1 et b_2 est disjoint avec b_1 dans l'ontologie O_2 et vice versa.
c) Un mapping propriété-propriété (p_1, p_2) est inconsistant en respectant l'alignement A si $\{Doms(p_1) \text{ X } Doms(p_2)\} \cap A = \emptyset$; et $\{Rans(p_1) \text{ X } Rans(p_2)\} \cap A = \emptyset$, où Doms(p) et Rans(p) retournent un ensemble de domaines et de co-domaines (ranges) d'une propriété p.

2.2.4. La fusion des classes similaires

L'étape de fusion des classes similaires est principalement basée sur la liste finale des mappings valides pour que chaque couple de classes candidates soit fusionné en une seule classe. Pour ce faire, notre système est, d'une part, basé sur un mécanisme de préférence. En effet, notre système initialise l'ontologie résultat par l'ontologie préférée. Pour des raisons de simplification, notre système considère l'ontologie la plus grande (celle ayant le plus grand nombre de classes) comme étant l'ontologie préférée. Ce mécanisme permet à notre système de résoudre automatiquement la plupart des conflits pouvant être rencontrés lors de la fusion, en suivant les sémantiques de l'ontologie préférée. Par exemple, lors de la fusion des deux classes similaires (c_1 : *marchandise*) ayant une propriété p_1= *quantité* de type « *float* » et (c_2 : *marchandise*) ayant une propriété p_2= *quantité* de type « *double* ». A ce stade, pour résoudre automatiquement, le conflit rencontré lors de la fusion des deux propriétés (p_1 : *quantité* : *float* et p_2 : *quantité* : *double*), et en supposant que la première ontologie est l'ontologie préférée (l'ontologie la plus grande), alors, notre système va retenir la sémantique fournie par cette ontologie et donc va préserver le type « *float* » à la propriété fusionnée « *quantité* ». Le mécanisme basé sur la préférence ne permet pas uniquement de résoudre automatiquement la plupart

des conflits rencontrés lors de la fusion, mais il distingue notre système de la plupart des systèmes de fusion d'ontologies tel que PROMPT qui demande le feedback de l'utilisateur pour résoudre toutes les hétérogénéités et/ou conflits. Ceci, constitue un vrai challenge lors de la fusion des ontologies de grandes tailles.

Une fois que la liste finale des mappings valides est diffusée au processus de fusion, ce dernier commence l'exécution de l'opération de fusion qui consiste en deux opérations principales : fusionner les classes similaires et copier les classes dissimilaires. L'opération de la fusion est exécutée de telle façon qu'elle unifie toutes les informations contenues dans les ontologies sources en consommant le moins de ressources machine possibles, i.e. la mémoire de stockage et le temps d'exécution. C'est pour cette raison que notre algorithme fusionne une ontologie source dans une autre. Ceci facilite l'obtention de toutes les connaissances de la première ontologie dans l'ontologie résultat o_3. Ensuite, il y ajoute la deuxième ontologie o_2 pour générer l'ontologie fusionnée complète o_3.

Donc, après l'importation des deux ontologies sources o_1 et o_2 ainsi que l'obtention de la liste finale des mappings valides, notre algorithme de fusion commence par initialiser l'ontologie résultat par l'ontologie préférée. Ensuite, et en parcourant les deux ontologies niveau par niveau, de haut en bas et de gauche à droite, il exécute la fusion des classes du premier niveau de la deuxième ontologie (les sous classes de la classe universel *thing*). À ce stade, notre algorithme vérifie également les mappings entre propriétés (et instances) des classes fusionnées. Ensuite, il fusionne les propriétés et instances similaires et copie les propriétés et instances différentes. Ici, une classe unique c_3 qui est décrite par toutes les propriétés et instances des deux classes c_1 et c_2 sans duplication, est générée à partir de la fusion des deux classes c_1 et c_2. À ce stade, deux types de problèmes majeurs sont résolus :

1. La perte d'informations : Toutes les informations contenues dans les deux ontologies sources o_1 et o_2 sont préservées dans l'ontologie résultat o_3.

2. La duplication ou le chevauchement d'informations : Par exemple, si une propriété est contenue dans les deux classes c_1 et c_2, elle va être sauvegardée une seule fois dans la classe c_3.

Dans le cas où une classe n'a pas de classes similaires (selon la liste finale des mappings valides), elle sera copiée directement (avec ses propriétés et instances) dans l'ontologie résultat sous son super classe à son origine (dans la hiérarchie de la deuxième ontologie). Le résultat de cet algorithme est la hiérarchie de l'ontologie fusionnée.

La partie de l'algorithme chargée de la construction de l'ontologie résultat est donnée par le pseudo code suivant :

Entrées : Les ontologies sources o_1 et o_2 en OWL-DL.
L'ontologie préférée, considérer l'ontologie la plus grande comme étant l'ontologie préférée.
La liste finale des mappings valides ordonnée par les niveaux de classes de la deuxième ontologie (qui n'est pas préférée).
Sorties : L'ontologie résultat de la fusion o_3, en OWL-DL.
Algorithme :

1. Initialiser l'ontologie o_3 par l'ontologie préférée o_1.
2. Copier les classes c_2 du niveau 1 de l'ontologie o_2 dans un vecteur V.
3. Exécuter les instructions 4 et 5 sur toutes les classes du vecteur V, jusqu'à sa fin.
4. Pour chaque classe c_2 du vecteur V, chercher si elle a une classe similaire c_1, à partir de la liste des mappings valides.
   Si une telle classe similaire c_1 existe alors
   - Fusionner les deux classes c_1 et c_2 dans o_3 (vu que c_1 et c_2 sont similaires, c_1 va prendre le nom de c_2, pour des raisons de simplification)
   - Fusionner leurs propriétés et instances similaires
   - Copier leurs propriétés et instances dissimilaires

   Sinon, si une telle classe similaire c_1 n'existe pas alors
   Si c_2.niveau =1 alors copier c_2 dans o_3 sous la classe universel, *thing*
   Sinon, si c_2.niveau<>1 alors copier c_2 dans o_3 sous sa classe père dans l'ontologie o_2.
5. Ajouter toutes les sous classes de la classe c_2 à la queue du vecteur V.
6. Si V est vide alors break
7. O_3 est l'ontologie résultat de la fusion.

2.2.5. Vérification et test de l'ontologie issue de la fusion

Une fois que l'ontologie issue de la fusion est générée, il est fortement recommandé de vérifier son exactitude et sa précision. Il est possible que l'ontologie résultat de la fusion contienne des inconsistances, des redondances et /ou des incomplétudes. Plusieurs outils d'évaluation et/ou moteurs de raisonnement ontologiques basés sur la logique de

description, existent dans la littérature. Nous avons choisi d'utiliser le moteur de raisonnement ontologique basé sur la logique de description, Racer++, pour vérifier la consistance, la redondance ainsi que la complétude de nos ontologies issues de la fusion.
Notre système est enfin testé et appliqué pour fusionner des ontologies du monde réel, telles que des ontologies du benchmark de la campagne *OAEI'2015*, des ontologies industrielles représentant une turbine à vapeur, des ontologies représentant le département d'informatique, etc. (Voir chapitre 5).

3. Comparaison conceptuelle d'AOM-FOM avec d'autres algorithmes de fusion

Dans cette section, nous allons comparer notre système de fusion automatique d'ontologies, AOM-FOM avec des systèmes de fusion déjà existant dans la littérature. Nous avons comparé notre système avec trois catégories différentes de systèmes d'alignement et /ou de fusion d'ontologies. La première catégorie contient des algorithmes d'alignement et/ou de fusion d'ontologies semi-automatiques. La deuxième catégorie contient des algorithmes d'alignement et/ou de fusion d'ontologies complètement automatiques. Alors que la troisième catégorie contient les algorithmes d'alignement et/ou de fusion d'ontologies dont les ontologies de test sont téléchargeables à partir du web et les résultats analytiques obtenus sont également consultables sur le site web de la campagne *OAEI*
La comparaison effectuée dans cette section est une comparaison conceptuelle. Une autre comparaison analytique des résultats (performances) obtenus par notre système AOM-FOM avec la troisième catégorie des systèmes d'alignement et/ou de fusion d'ontologies est présentée dans la deuxième partie du chapitre 5 (voir Chap. 5 §6.1.). Alors, en comparant les performances conceptuelles attendues par notre algorithme AOM-FOM avec les algorithmes de fusion les plus connus et les plus cités dans la littérature tels que PROMPT, PROMPT-Diff, PROMPT-Anchor, Chimaera, ONION, FCA-Merge, Glue, OntoMerge, HCone-Merge, etc., nous notons que parmi les points forts de l'algorithme proposé est qu'il est complètement automatique, on n'a besoin d'aucune intervention

humaine pour choisir et/ou valider les mappings candidats à la fusion. Il est basé sur un algorithme automatique et flexible de mapping où les différentes techniques de mappings sont appliquées selon le contenu et la structure des ontologies sources. Il utilise plusieurs stratégies différentes. Si une stratégie n'arrive pas à détecter des mappings, une autre stratégie va accomplir la tâche. Le point le plus important est que la liste initiale des mappings détectés est passée par un module de validation avant d'être diffusée au module de la fusion. Cette importance est liée au fait que la validation des mappings initiaux permet d'éliminer les mappings ayant des conflits et alors, éviter la génération d'une ontologie fusionnée erronée.

Le tableau 4.1 résume la comparaison de notre algorithme avec les algorithmes de la première catégorie (les algorithmes semi-automatiques). Alors que le tableau 4.2 résume sa comparaison avec les algorithmes de la deuxième catégorie (les algorithmes de fusion complètement-automatiques). Et le tableau 4.3 présente sa comparaison avec les algorithmes de la troisième catégorie (les algorithmes dont les ontologies de test ainsi que les résultats obtenus sont téléchargeables à partir du site web de la compagnie OAEI'2015).

Table 4.1. Comparaison d'AOM-FOM avec les algorithmes de la première catégorie

Propriétés	Chimaera	ONION	PROMPT	FCA-Merge	Glue	Prompt-Diff	Prompt-Anchor	OntoMerge	Hcone-Merge	AOM-FOM
Automation	Semi-automatique	Semi-automatique	Semi-automatique	Semi-automatique	Semi-automatique	Semi-automatique	Semi-automatique	Semi-automatique	Semi-automatique	Complètement automatique
Opération	Fusion	Composition	Mapping + fusion	Fusion	Mapping	Comparaison de versions d'ontologies	Mapping	Fusion + raisonnement + translation	Fusion	Mapping + Fusion
Cadre de travail	Ontolingua	Indépendant	Protégé 2000	Indépendant	Indépendant	Protégé 2000	Protégé 2000	Indépendant	Indépendant	Indépendant
Langage de représentation	Ontologua	Graphes libellée et orientés + règles de Horn	Rdfs + owl	Taxonomies de classes d'ontologies populaires	Taxonomies	Taxonomies	Rdfs + owl	DAML : ontologies sources et fusionnée. Web pddl : raisonnement automatique	Logique de description	OWL-DL
Langage de mapping	Mesure de similarité linguistique	Mesure de similarité linguistique. Inférence structurelle basée sur les heuristiques. Règles d'articulation	Analyseur basé heuristiques	Mesure de similarité linguistique + algorithme TITANIC pour calculer le treillis de classes élaguées	Approche d'apprentissage multi-stratégies 'technique d'apprentissage automatique	Mapping basé heuristiques	Score de similarité structurelle	/	s-morphismme	Mesure de similarité linguistique + Apprentissage automatique + Inférence structurelle basée sur les heuristiques

Ressources externes	Aucun	wordNet	Aucun	Aucun	Aucun	Aucun	Aucun	Aucun	wordNet	wordNet
Analyse lexicale	Non	Non	Oui	Oui	Oui	Oui	Oui	*	Oui	Oui
Analyse sémantique	Oui	Oui	Oui	Oui	Oui	Non	Oui	*	Oui	Oui
Analyse de Classes	Non	Oui	Oui	Non	Oui	Oui	Oui	*	Oui	Oui
Analyse de Propriétés	Non	Non	Non	Non	Oui	Oui	Non	*	Non	Oui
Analyse de Instances	Non	Non	Non	Oui	Oui	Oui	Non	*	Non	Oui
Analyse de Structure	Non	Oui	Oui	Oui	Oui	Oui	Oui	*	Oui	Oui
Interaction avec l'utilisateur humain	Prendre des décisions affectant le processus de la fusion	L'expert humain choisit, supprime ou modifie les mappings proposés en utilisant un outil GUI	Accepter, rejeter ou ajuster les suggestions du système	Choisir la fonction de calcul de similarité	Définir les mappings pour l'ensemble d'apprentissage. Choisir la fonction de calcul de similarité. Fixer les poids du classifieur. Analyser les mappings suggérés	Examiner et confirmer ou rejeter les mappings basés sur la combinaison d'heuristiques	L'utilisateur peut (pas obligatoirement) définir l'ensemble des Anchors initiaux	Générer manuellement, l'ontologie fusionnée qui combine les l'ontologie source DAML avec sa nouvelle version translatée en ontologie DAML	Décider et choisir les stratégies de fusion. Guider le processus en cas d'inconsistance. Valider les sens prétendus des termes.	Aucune

					par le classifieur.					
Entrées	2 fichiers en RDF et DAML	Fichier RDF	2 ontologies daml (OWL)	2 ontologies en DL + ensemble de documents	2 taxonomies de classes + instances	2 versions différentes de la même ontologie	Paires d'Anchors	Une ontologie source en DAML	2 ontologies sources	2 ontologies sources en OWL-DL
Sorties	Ontologie fusionnée	Ensemble de règles d'articulation entre 2 ontologies	Ontologie fusionnée	Ontologie fusionnée	Ensemble de paires de classes similaires	Table de différences structurelles entre 2 versions d'une ontologie	Nouveaux paires avec les similarités correspondantes	Ontologie fusionnée et translatée	Ontologie fusionnée	Une ontologie fusionnée en OWL-DL
Architecture de mapping	Point à point Top-down	Point à point Top-down	Point à point Top-down	Point à point Bottom-up	Point à point Top-down	Point à point Top-down	Point à point Top-down	Selon l'expert humain	Point à point Bottom-up	Point à point Top-down Gauche-droite
Algorithme/ API utilisé	/	SKAT	OKBC	Classe lattice	Coefficient de Jaccard + MSP	Combinaison d'heuristiques	Plugin sous protégé 2000	Rien pour la fusion + raisonnement à base de chaînage avant et arrière pour la translation	Nee classic DL	Jena OWL-API Jwnl MorphAdorner Sim-Metrics Weka
Apprentissage	Non	Non	Oui	Non	Oui	Oui	Non	Non	Non	Oui

automatique										
Développé par	Université de Stanford, USA, 2005	Université de Stanford, USA, 2004	Stanford medical Informatics, Université Stanford, USA, 2003	Université de Karlsruhe, Germany, 2001	Université de Wisconsin, 2005	Université de Stanford, USA, 2003	Université de Stanford, USA, 2003	DARPA/DA ML program, USA , 2006	AI lab, greece, 2002	Université Annaba, Algérie, 2016

* : Oui ou non, ça dépend des experts humains, car la fusion est purement manuelle dans ces cas.

Table 4.2. Comparaison d'AOM-FOM avec les algorithmes de la deuxième catégorie

Propriétés	Algorithme de (Robin & Uma, 2010b)	Algorithme de (Maiz et al., 2010)	Algorithme de (Cho, Kim, et al., 2006b)	Algorithme de (S. Li et al., 2009)	Algorithme de (Mohsenzadeh et al., 2005)	Algorithme de (Kong et al., 2005)	AOM-FOM
Automation	Automatique	Automatique	Automatique	Automatique	Automatique	Automatique	Automatique
Opération	Fusion	Fusion	Fusion	Fusion	Fusion	Fusion	Mapping + Fusion
Cadre de travail	Indépendant	Indépendant	Indépendant	Indépendant	Indépendant	Indépendant	Indépendant
Langage de représentation	OWL	Taxonomies	Taxonomies	Taxonomies	Taxonomies	Taxonomies	OWL-DL
Langage de mapping	Mesure de similarité + fonction d'heuristiques	Mesure de similarité + relations fils-père	Mesure de similarité	Mesure de similarité	Mesure de similarité	Mesure de similarité	Mesure de similarité linguistique + Inférence

								structurelle basée sur les heuristiques + Apprentissage automatique
Ressources externes		WordNet	Aucun	WordNet	Aucun	WordNet	WordNet	wordNet
Analyse lexicale		Oui	Oui	Non	Non	Oui	Non	Oui
Analyse sémantique		Oui	Oui	Oui	Oui	Oui	Oui	Oui
Analyse de	Classes	Oui	Oui	Oui	Oui	Oui	Oui	Oui
	Propriétés	Oui	Non	Non	Non	Non	Oui	Oui
	Instances	Oui	Non	Non	Oui	Non	Oui	Oui
	Structure	Oui	Oui	Oui	Oui	Non	Oui	Oui
Interaction avec l'utilisateur humain		Aucune	Aucune	Aucune	Aucune	Aucune	Aucune	Aucune
Entrées		Deux ontologies	Plusieurs (≥2) ontologies	Deux ontologies	Plusieurs (≥2) ontologies	Plusieurs (≥2) ontologies	Deux ontologies	2 ontologies sources en OWL-DL
Sorties		Ontologie fusionnée	Ontologie fusionnée	Ontologie fusionnée	Ontologie fusionnée	Ontologie fusionnée	Ontologie fusionnée	Une ontologie fusionnée en

							OWL-DL
Architecture de mapping	Point à point Bottom-up	Ce n'est pas applicable	Techniques horizontale et verticale	Point à point Top-down	Point à point Top-down	Point à point Top-down	Point à point Top-down
Algorithme/ API utilisé	*	Algorithme de classification hiérarchique + inférence	*	Algorithme de similarité d'ontologies + algorithme de construction de treillis de classes	Algorithme de programmation dynamique	*	Jena OWL-API Jwnl MorphAdorner Sim-Metrics Weka
Apprentissage automatique	Non	Non	Non	Non	Non	Non	Oui
Développé à	Université Anna, Jerusalem, India, 2010	Lab ERIC, université lumière, France, 2010	Université Che Ju, Chosum, Korea, 2010	Université Hubei, Wuhan, China, 2009	Université KNT, Islamic Azad, Iran, 2005	Université Chosun, Ersity, Korea, 2005	Université Annaba, Algérie, 2016

* : Inconnu

Table 4.3. Comparaison d'AOM-FOM avec les algorithmes de la troisième catégorie

Propriétés	Yam++	DKP-AOM	LogMap	XMap++	AOM-FOM
Automation	Semi-Automatique	Automatique	Semi-Automatique	Automatique	Automatique
Opération	Alignement	Fusion	Alignement	Alignement	Mapping + Fusion
Cadre de travail	Indépendant	Indépendant	Indépendant	Protégé2000	Indépendant
Langage de représentation	OWL	OWL	OWL	OWL	OWL-DL
Langage de mapping	Mesure de similarité + Propagation de similarité	Mesure de similarité	Mesure de similarité	Mesure de similarité + Réseaux de neurones artificiels	Mesure de similarité linguistique + Inférence structurelle basée sur les heuristiques + Apprentissage automatique
Ressources externes	WordNet +Microsoft Bing	WordNet	WordNet	WordNet	wordNet

Analyse lexicale		Oui	Oui	Oui	Oui	Oui
Analyse sémantique		Oui	Oui	Oui	Oui	Oui
Analyse de	Classes	Oui	Oui	Oui	Oui	Oui
	Propriétés	Oui	Oui	Oui	Oui	Oui
	Instances	Oui	Non	Oui	Oui	Non
	Structure	Oui	Oui	Oui	Oui	Oui
Interaction avec l'utilisateur humain		Oui	Aucune	Oui	Aucune	Aucune
Entrées		Deux ontologies	Deux ontologies	Deux ontologies	Deux ontologies	2 ontologies sources en OWL-DL
Sorties		Liste de mappings	Ontologie fusionnée	Liste de mappings	Liste de mappings	Une ontologie fusionnée en OWL-DL
Architecture de mapping		Point à point Top-Down	Top-Down Left-most	Top-Down	Point à point Top-down	Point à point Top-down
Algorithme/ API utilisé		OWL-API Top-K algorithm	OWL-API + Jena + MorphAdorner	Dowling-Gallier	OWL-API + Jena +weka	Jena OWL-API Jwnl

		API + SimMetrics API			MorphAdorner Sim-Metrics Weka
Apprentissage automatique	Oui	Non	Non	Oui	Oui
Développé à	Lab. LIRMM. Université Montpellier, France, 2013	Lab. CERRAL, université lumière, France, 2015	Université d'Oxford, UK, 2015	Université Annaba, Algérie, 2015	Université Annaba, Algérie, 2016

4. Conclusion

La fusion d'ontologies joue un rôle technique crucial pour favoriser et supporter l'interopérabilité entre différents systèmes. Dans ce chapitre, nous avons présenté et expliqué l'algorithme de notre système tout en justifiant nos choix de solutions. AOM-FOM décrit un outil de fusion automatique d'ontologies qui utilise un algorithme flexible de mapping. Ce dernier est basé sur la combinaison de plusieurs stratégies de mapping. Si une stratégie échoue à détecter des mappings, une autre va accomplir la tâche. Notre système combine alors les mappings détectés au niveau structurel (qui détecte des mappings entre les classes et leurs propriétés) avec les mappings détectés au niveau entité. A leur tour, les mappings détectés au niveau entité sont les résultats de la combinaison des mappings détectés au niveau schéma des ontologies (qui détecte également, des mappings entre les classes et leurs propriétés) et les mappings détectés au niveau extensionnel des ontologies (qui détecte des mappings entre les classes en se basant sur leurs instances). La stratégie de détection des mappings au niveau extensionnel est basée sur l'apprentissage automatique des instances en utilisant la technique des arbres de décision. Alors que toutes les autres stratégies sont basées sur l'analyse lexicale et/ou sémantique des chaînes de caractères étiquetant les classes et leurs propriétés. FOM est doté par un module de validation pour accepter ou rejeter les mappings de la liste initiale et obtenir la liste finale des mappings.

Parmi les points forts de notre algorithme, AOM-FOM est qu'il est complètement automatique. Il ne demande aucune intervention humaine pour l'extraction de mappings initiaux et/ou leurs validations. La flexibilité de l'algorithme de mapping nous permet d'appliquer notre système sur des ontologies de structures et de contenus variés. Et le module de validation automatique des mappings rejette les mappings erronés et permet alors de générer une ontologie fusionnée complète, consistante et cohérente.

Dans le chapitre suivant, nous allons montrer l'implémentation et le test de l'efficacité de notre approche pour fusionner des ontologies réelles tout en comparant les performances obtenues avec celles des algorithmes de fusion qui existent dans la littérature.

Chapitre 5 . Implémentation du système AOM-FOM

1. Introduction

Dans la première partie de ce chapitre, nous allons démontrer l'implémentation de notre système de fusion automatique d'ontologies basé sur l'alignement flexible d'ontologies sources, (AOM-FOM, Automatic Ontology Merging based on a Flexible Ontology Matching). D'abord, nous survolons les différentes APIs que nous avons utilisées. Ensuite, nous présentons une démonstration de notre algorithme tout en discutant les différents services et fonctionnalités de notre framework de fusion automatique d'ontologies hétérogènes.

Dans la deuxième partie de ce chapitre, nous allons présenter les résultats expérimentaux de notre système en évaluant ses différents approches et algorithmes. Nous montrons, dans un premier lieu, un exemple concret de l'influence de la prise en compte des partitions disjointes sur la réduction du nombre de comparaisons effectuées pour chercher les mappings candidats à la fusion. Nous appliquons en second lieu notre système AOM-FOM pour fusionner quelques ontologies du benchmark *OAEI'2015* (*Ontology Alignement Evaluation Initiative' 2015*), tel que les ontologies du track *biblio* (sous track du track *benchmark*) décrivant le domaine de la gestion des références bibliographiques, en plus des ontologies du track *conference* représentant le domaine de la gestion des conférences, tout en comparant les résultats obtenus par notre système avec ceux obtenus par quelques systèmes d'alignement et de fusion qui ont participé dans la campagne *OAEI' 2015*). Puis, notre système est appliqué pour fusionner des ontologies de différents domaines tels que le domaine d'organisation de conférences, le domaine industriel représentant les turbines à vapeur[5] en plus des ontologies représentant le département d'informatique. Finalement, et en nous basant sur les expérimentations menées, nous discutons la fiabilité de notre système.

[5] Automate utilisée pour la génération de l'électricité.

2. Partie I : Implémentation du système AOM-FOM

2.1. Les différentes APIs utilisées pour l'implémentation d'AOM-FOM.

Dans la section suivante, nous allons introduire les différentes APIs que nous avons utilisées pour l'implémentation de notre système de fusion automatique d'ontologies sources, AOM-FOM, tout en expliquant comment nous les avons utilisées.

2.1.1. *Jena*, un Framework pour le web sémantique.

Jena est le Framework le plus populaire pour le Web sémantique sous Java. Il est développé sous le programme du web sémantique des laboratoires HP (Carroll et al., 2003). Il supporte un environnement de programmation riche pour *RDF, RDFS, OWL, SPARQL*, etc. *Jena* est utilisé pour construire un graphe d'une ontologie dans la mémoire pour le traitement rapide des entités ontologiques. Nous avons utilisé la version 2.5 du framework *Jena* pour mettre en œuvre notre framework de fusion automatique d'ontologies *OWL* (*AOM-FOM*). Bien que *Jena* permette d'écrire l'ontologie fusionnée sous la syntaxe *RDF, N3 et Turtle*, il ne fournit pas un analyseur *OWL / XML* ni un éditeur pour écrire le modèle d'ontologie sous la syntaxe *OWL*, il est donc impossible de produire une ontologie fusionnée sous le format *OWL*. C'est pourquoi, nous sommes passée à l'API *OWL-API* pour la production d'une ontologie fusionnée sous la syntaxe *OWL*.

2.1.2. OWL-API, Une API pour le web sémantique.

De nos jours, *OWL-API* est largement utilisée pour développer des applications pour le web sémantique vu ses capacités de traiter et de manipuler les constructions riches de *OWL*. Il est spécialement conçu pour créer, manipuler et sérialiser les ontologies *OWL*. Nous avons utilisé la version 3.0 de l'API *OWL-API* développée par Horridge et Bechhofer (2011) pour le développement de notre système de fusion. Notre système est alors capable d'écrire le modèle d'ontologie fusionnée sous forme d'un fichier de syntaxe *OWL* mais aussi sous forme de différentes autres syntaxes d'ontologie. Nous

avons trouvé *qu'OWL-API* est une API web sémantique très puissante pour manipuler des ontologies *OWL* avec leurs constructeurs riches.

2.1.3. *MorphAdorner*, Lemmatisation des entités ontologiques.

MorphAdorner est un programme Java utilisé pour effectuer des traitements morphologiques sur les mots dans le texte. Il a été développé en 2009 par l'Université *Northwestern* pour être utilisé dans le projet *Monk*, où le nombre total de mots traités était d'environ 151 500 000. Il fournit des méthodes riches pour le traitement du langage naturel et enrichit le texte avec des orthographes standards, détecte les formes de base (lemma), reconnait les limites des phrases, et extrait les noms et leurs positions. Nous avons utilisé *MorphAdorner* pour le pré-traitement des termes ontologiques avant de trouver les mappings candidats entre termes. Les deux tâches très importantes, *stemming* et *lemmatisation* sont vraiment utiles dans ce contexte.

Le stemming réduit un mot à sa forme de base en supprimant les affixes, mais le stem qui en résulte n'est pas nécessairement un lemme propre ou une forme de base. Par conséquent, nous avons utilisé le service lemmatisation qui entraîne une forme de base correcte.

La lemmatisation réduit une "orthographe fléchie" (inflected spelling) (c.à.d. sous une forme conjuguée et non pas sous une forme de base) vers sa forme de racine lexicale ou lemme, qui est une forme de base du mot. Pour les noms de classes, il les réduit à leur forme de base, c'est à dire, pour un nom, il le réduit à sa forme singulière. Par exemple, il réduit "*souris*" au pluriel à sa forme singulière (lemme) "*souris*", et «*ministères*» à sa forme singulière (lemme) «*ministère*». Les Propriétés d'objet ou encore les propriétés de type de données sont la plupart du temps des verbes, comme *"écrire"*, *"écrite"*, etc. Il permet alors de convertir ces termes à leurs formes de base qui est pour un verbe l'infinitif de l'indicatif. Par exemple, les formes *"écrit"*, *"a écrit"*, *"écrirait"*, *"écrivant"*, *etc.*, sont des formes variées du lemme *"écrire"*. Dans notre système, nous avons utilisé ce service principalement pour détecter des mappings entre les propriétés, vu que la plupart du temps elles sont dans leur forme verbale (par exemple, *écrit*, *lit*, *enregistre*, *etc*.) et / ou concaténé avec certains verbes ou mots

d'aide (par exemple, *écritPar, est-enregistré, envoyéPour, etc.*). La lemmatisation joue un rôle important lors de l'alignement des propriétés.

2.1.4. *SimMetrics* – Similarity Mertics

C'est une bibliothèque de programmes java, open source, pour mesurer la similarité entre chaînes de caractères. Elle est développée par Chapman (2007) du groupe de traitement de la langue naturelle à l'université *Sheffield*. L'API *SimMetrics* est alors développée pour implémenter des tâches d'intégration d'information. Elle est équipée, presque de tous les algorithmes les plus importants pour mesurer la similarité entre chaînes de caractères tels que *la distance d'Edition, la distance de Levenshtein, la similarité de Cosine, le coefficient de Jaccard, la distance de Jaro, le coefficient de Dice, la distance Qgrame, la distance de Jaro Winkler, la distance de Monge Elkan, la distance de Needlemaan Wunch, la distance de Smith Waterman, la distance de TagLink, le coefficient Overlap, etc.*

Ces algorithmes sont développés et exploités par différentes communautés et dans différents domaines de recherche tels que l'Intelligence Artificielle, les Statistiques, les Bases de données, l'analyse de l'ADN ou la Recherche d'Informations. L'intégration de *SimMetrics* dans *AOM-FOM* donne encore une fois un autre type de flexibilité à l'utilisateur final pour choisir l'algorithme de mesure de distance entre chaînes de caractères (pour détecter les mappings basés sur les chaînes de caractères entre entités ontologiques) qui lui convient le plus, selon le domaine d'application des ontologies sources.

2.1.5. WordNet

C'est une base de données lexicale anglaise. La technique de découverte de mappings basée sur les synonymes est principalement fondée sur *WordNet* ou encore *WordNet Princeton*. Il s'agit d'une ontologie pour la langue anglaise qui contient des termes, leurs sens (ou synonymes), leurs définitions ainsi que leurs relations avec d'autres termes. Le projet *WordNet* a commencé en 1995 à l'université de Princeton et actuellement, il est très bien établi avec le vocabulaire de la langue anglaise. *WordNet* est aujourd'hui dans sa troisième version de mise à jour pour contenir de plus en plus de vocabulaire et de définitions anglaises. La structure de *WordNet* est principalement

basée sur les ensembles de synonymes appelés « *synset* ». Chacun des synsets représente les sens d'une classe. En d'autres termes, chacun d'eux contient l'ensemble de tous les synonymes pouvant exprimer les sens de la classe en question. La figure 5.1 représente trois *synsets* du mot « *department* ». En plus de ses définitions, la figure 5.2 représente les différents types de département à travers l'ensemble des hyponymes du terme « *department* ».

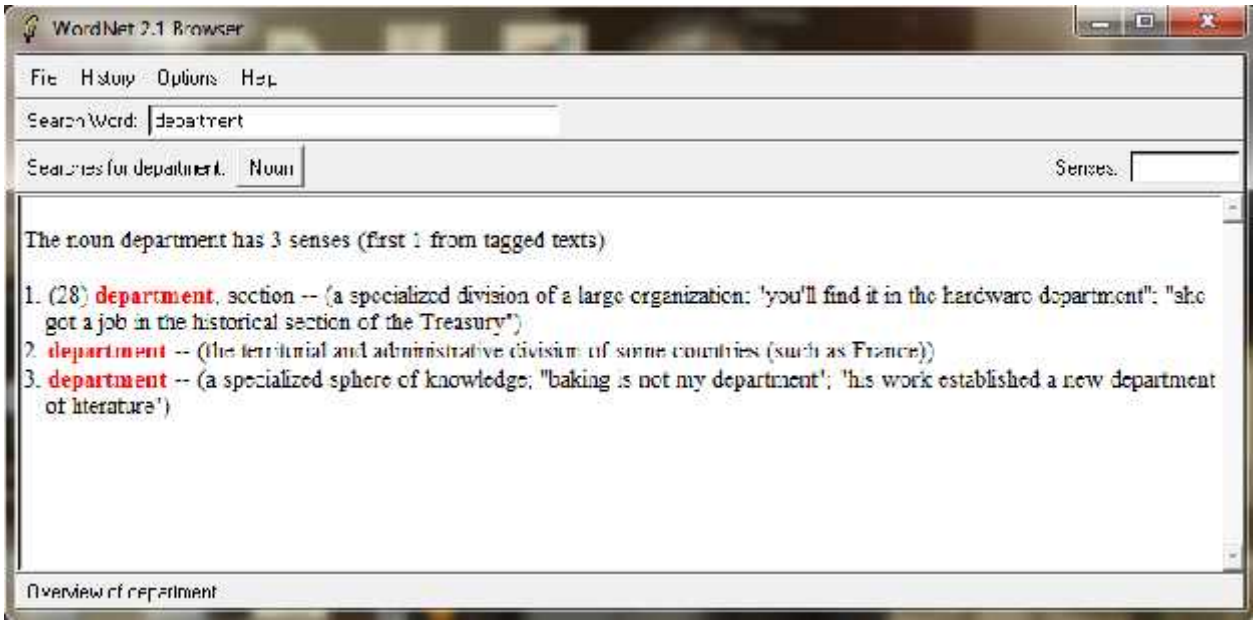


Figure 5-1. Les sens de la classe« department » par WordNet

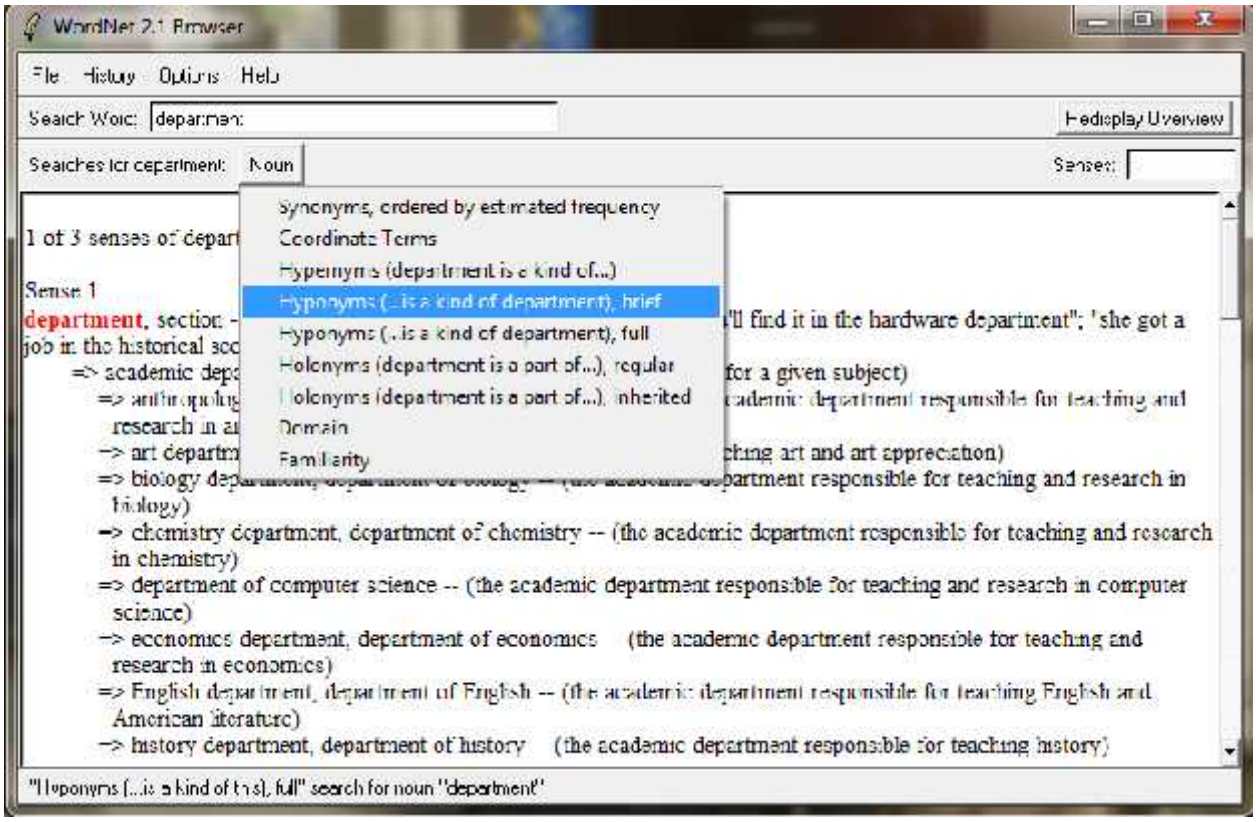


Figure 5-2. Les hyponymes de la classe« department » par WordNet

2.1.6. Weka -Waikato Environment for Knowledge Analysis-

C'est une bibliothèque de programmes open source d'apprentissage automatique. Elle est développée à l'université de Waikato en Nouvelle Zélande et écrite en Java. Elle se compose principalement: Des classes Java permettant de charger et de manipuler les données. Des classes pour les principaux algorithmes de classification (dont les arbres de décision). Des outils de sélection d'attributs et de statistiques sur ces attributs. Des classes permettant de visualiser les résultats. Weka peut être utilisé via une interface graphique, pour charger un fichier de données (sous format .arff) et lui appliquer un algorithme de classification ou encore l'invoquer sur la ligne de commande. L'importation de l'api Weka dans un programme java permet une exploitation aussi flexible que sa version graphique. Dans ce travail, nous avons utilisé l'api Weka pour construire le classifieur basé sur les arbres de décision pour classifier les classes (en classifiant leurs instances) de la deuxième ontologie avec leurs classes similaires de la première ontologie.

2.2. Description des interfaces de AOM-FOM

Dans cette section, nous allons présenter une description générale de l'organisation et/ou du positionnement de notre processus global de fusion d'ontologies. Nous allons introduire notre prototype AOM-FOM à travers une suite de screenshots d'exécution comme suit :

2.2.1. Une interface pour choisir l'ontologie préférée

Comme illustré par la figure 5.3, nous offrons une interface graphique afin que l'utilisateur final puisse spécifier l'ontologie préférée parmi l'une des deux ontologies sources, et alors l'importation des ontologies candidate à la fusion à notre processus de fusion automatique d'ontologies.

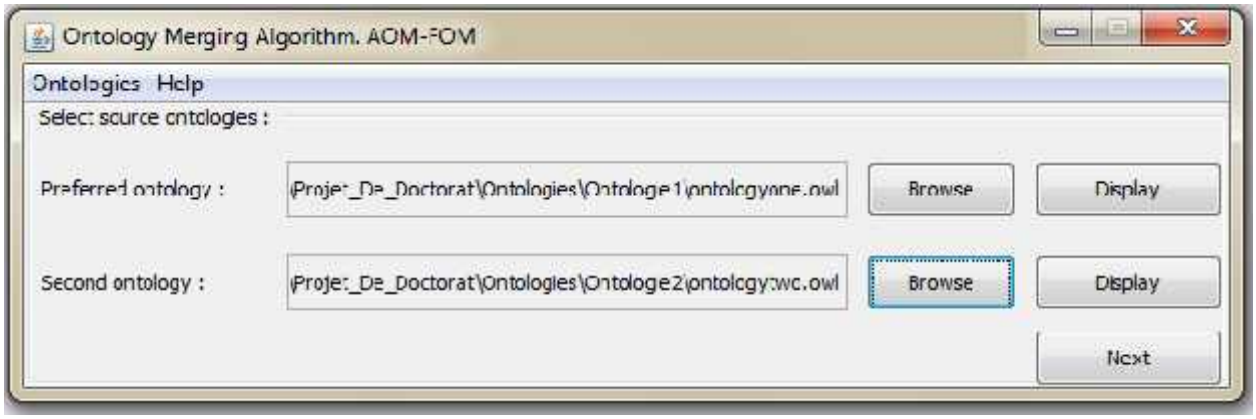


Figure 5-3. Une interface pour choisir l'ontologie préférée

2.2.2. Une interface de visualisation des deux ontologies sources sous forme d'une arborescence (Jtree)

Après l'importation des ontologies candidates à la fusion, notre système permet de les visualiser sous différents formats. Nous estimons que la forme d'arborescence est celle la plus pratique vue qu'elle facilite la compréhension et l'assimilation du processus global d'alignement et de fusion d'ontologies. La visualisation sous forme d'arborescence (tree view) permet à l'utilisateur final de visualiser les ontologies sources et/ou l'ontologie issue de la fusion sous forme de hiérarchie de classes. Elle lui permet également de visualiser les hiérarchies des ontologies sources ainsi que la hiérarchie de l'ontologie fusionnée sur la même fenêtre. Ceci lui facilite la compréhension du processus de fusion et /ou d'alignement en analysant les hiérarchies

des ontologies sources et ontologie fusionnée sur le même frame. La Figure 5.4 présente le screenshot visualisant les ontologies sources sous forme d'arborescence (tree view).

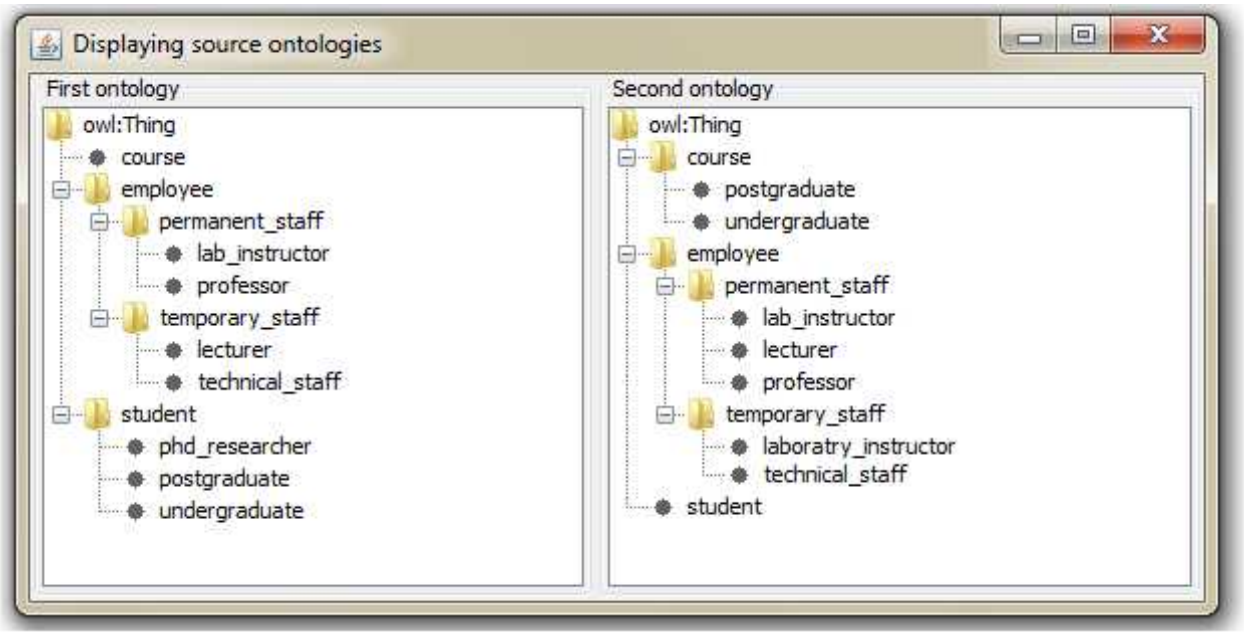


Figure 5-4. Une interface de visualisation des ontologies sources sous forme d'une arborescence (Jtree)

2.2.3. Des interfaces visualisant la liste des mappings candidats à la fusion

Une fois que le processus de calcul de similarité est exécuté, notre système permet de visualiser les résultats sous deux formes différentes. Une matrice de similarité (*matrix view*) ou une combinaison de la matrice de similarité associée avec les ontologies sources (*combined view*). La matrice de similarité affiche les valeurs de similarité calculées entre ontologies sources. Elle visualise alors toutes les correspondances entre les classes des ontologies sources ainsi que leurs valeurs de similarités. L'utilisateur final peut alors analyser et comprendre les correspondances entre ontologies sources. Alors que, l'affichage combiné affiche les ontologies sources ainsi que la matrice de similarité dans une seule fenêtre. Il permet alors une meilleure analyse et manipulation des ontologies sources ainsi que les résultats des mappings candidats à la fusion. La figure 5.5 et la figure 5.6 représentent, respectivement, la liste des mappings initiaux et

la liste des mappings finaux (valides) sous forme de matrices (matrix view) et la figure 5.7 représente la liste des mappings valides combinés par les hiérarchies des ontologies sources (combined view).

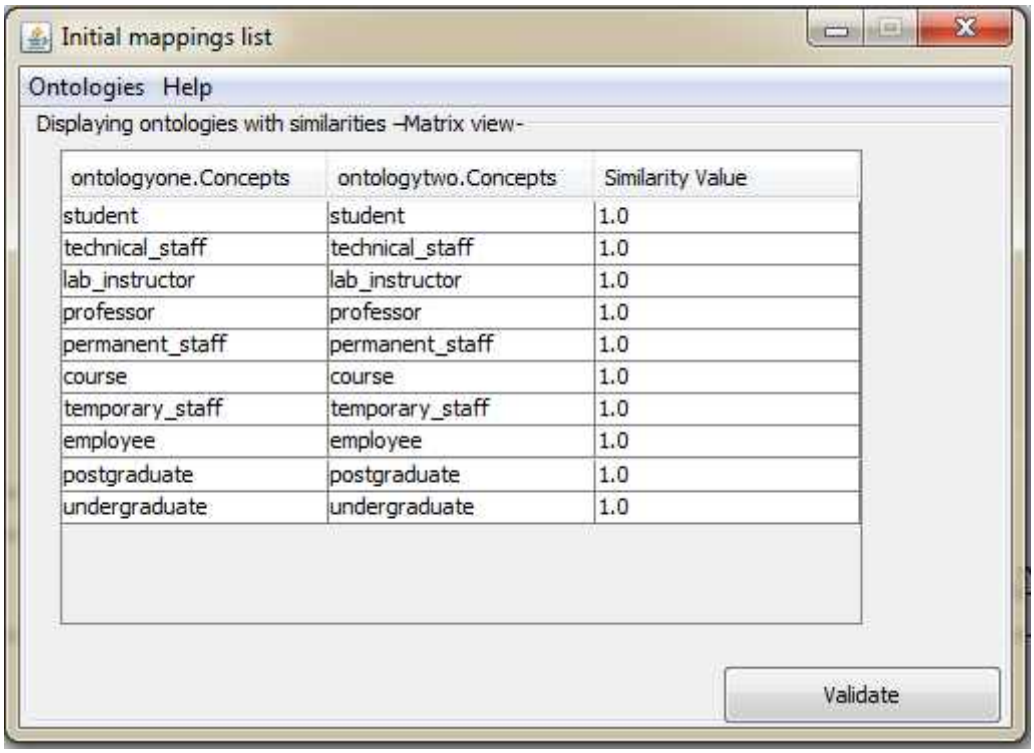


Figure 5-5 Une interface visualisant la liste initiale des mappings candidats à la fusion (matrix view).

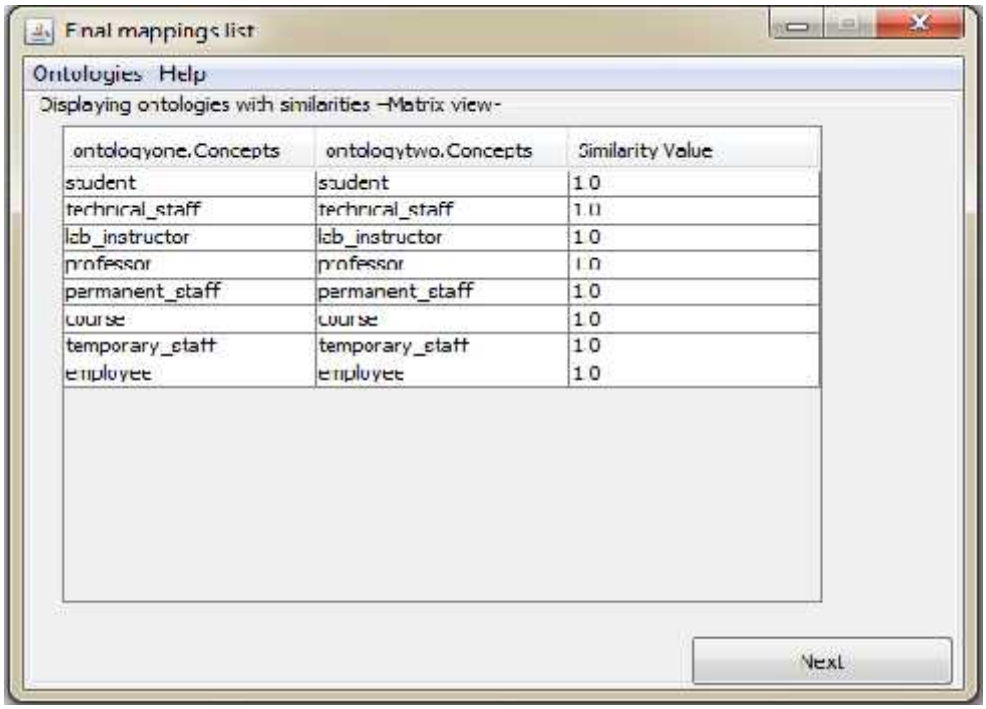


Figure 5-6. Une interface visualisant la liste finale des mappings candidats à la fusion (matrix view)

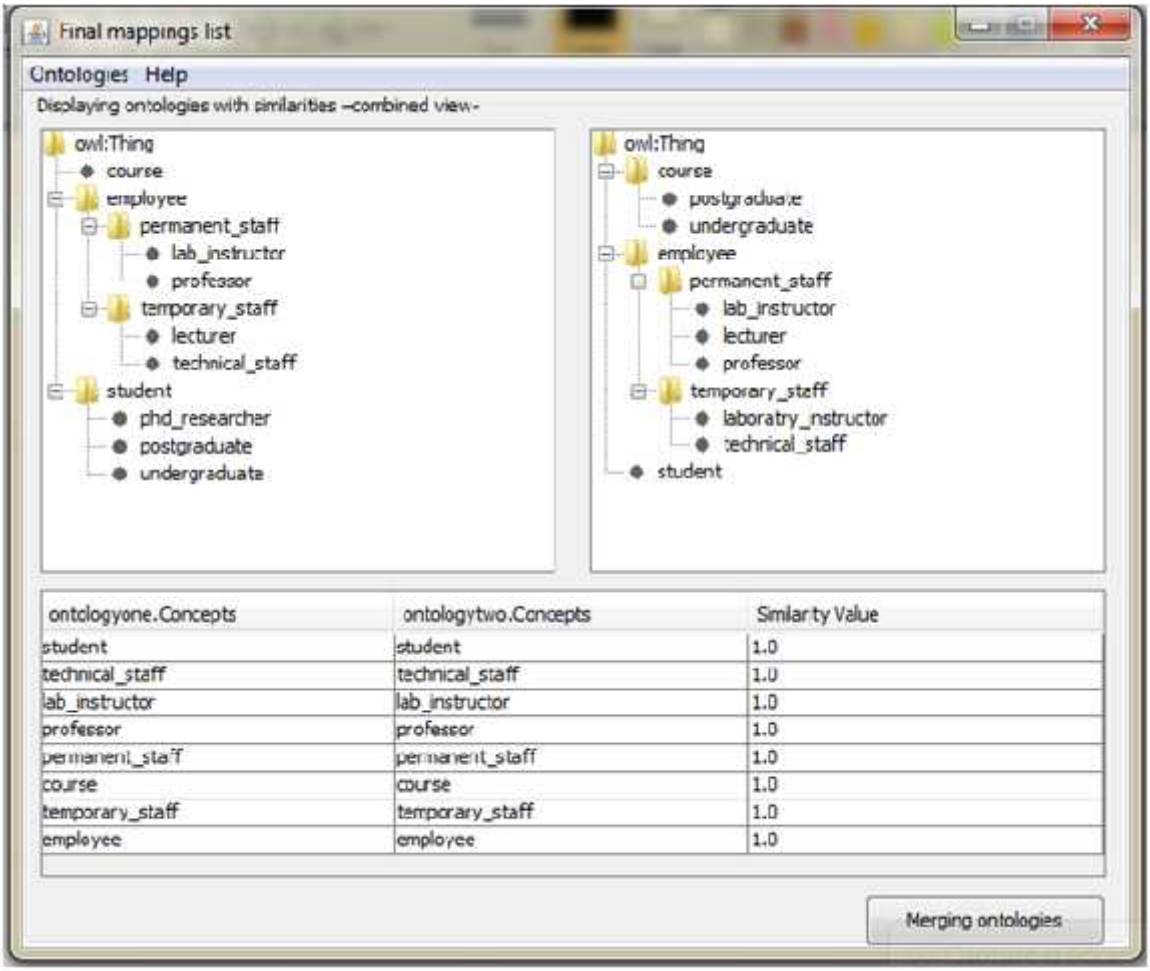


Figure 5-7. Une interface visualisant la liste finale des mappings candidats à la fusion combinée avec les hiérarchies des ontologies sources (combined-view)

2.2.4. Une interface pour visualiser l'ontologie issue de la fusion

Finalement, notre système affiche la hiérarchie de l'ontologie issue de la fusion sous une forme d'arborescence (*tree view*). (Voir la figure 5.8)

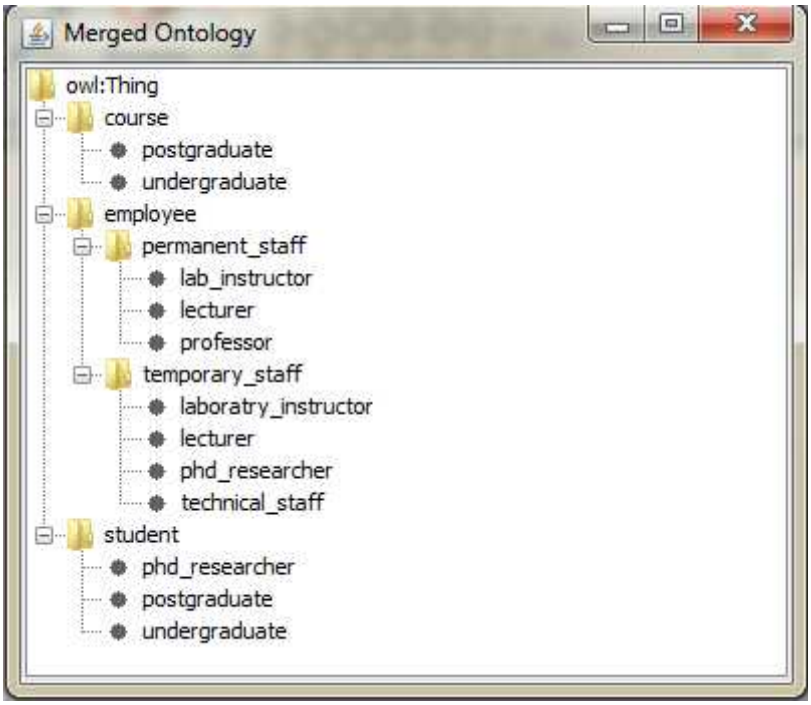


Figure 5-8. Une interface pour visualiser l'ontologie issue de la fusion (tree view).

2.3. Stratégies d'optimisation des ressources

Pour trouver les mappings candidats à la fusion à partir de deux ontologies sources o_1 et o_2 de taille n_1 et n_2, respectivement, on doit effectuer $n_1 * n_2$ comparaisons pour garantir de ne pas rater des mappings à fusionner. Cependant, et surtout quand on est devant des ontologies de grandes tailles, ceci peut consommer énormément de ressources machine (mémoire de stockage et temps d'exécution). Alors, pour réduire la consommation des ressources machine, l'analyse des partitions disjointes est une solution prometteuse. En effet, les axiomes de disjonction partagent les connaissances ontologiques en partitions disjointes, ce qui permet de réduire l'espace de recherche des mappings uniquement dans des sous hiérarchies au lieu de chercher dans toute la hiérarchie de l'ontologie.

Par exemple, dans les ontologies ***CMT*** et ***CRS-DR*** (du track *conference* de la compagne *OAEI'2015*), à partir des mappings :

- ***Mapping*** (*cmt : Person, crs-dr : Person) et*
- ***Mapping*** *(cmt : Document, crs-dr : Document*) et en utilisant l'axiome de disjonction :

- ***Disjoint-with*** *(Person, Document)*, alors : la recherche de mappings des sous classes de la classe ***cmt : Person*** doit s'effectuer dans la sous hiérarchie de la classe ***crs-dr : Person*** mais ni dans la sous hiérarchie de la classe ***crs-dr : Document*** ni dans la hiérarchie de l'ontologie ***crs-dr*** complète.

Alors, plus les axiomes de disjonction sont spécifiés dans les ontologies sources et plus ces ontologies sont libres des hétérogénéités conceptuelles, plus cette stratégie permet de réduire le nombre de comparaisons effectuées pour trouver les mappings candidats à la fusion. Ceci, permet de réduire significativement le temps d'exécution ainsi que l'espace mémoire utilisé pour trouver les classes candidates à la fusion.

Dans ce qui suit, nous allons tester l'efficacité de cette stratégie pour trouver les mappings candidats à la fusion des deux ontologies ***CMT*** et ***CRS-DR***. Ces deux ontologies sont très connues dans le domaine de gestion des conférences.

L'ontologie *CMT:* composée de **36** classes et **27** axiomes de disjonction sont spécifiés. Les axiomes de disjonction entre les classes du premier niveau (les sous classes de la classe *thing*) partagent les classes du domaine en six catégories disjointes (***Person, Decision, Document, Conference, Preference, et ProgramCommittee***).

L'ontologie *CRS-DR:* composée de **14** classes et **12** axiomes de disjonction sont spécifiés. Les axiomes de disjonction entre les classes du premier niveau partagent les classes du domaine en quatre catégories disjointes (***Program, Person, Document et Event***).

La figure 5.9 représente les partitions disjointes des classes du premier niveau des deux ontologies CMT et CRS-DR.

Figure 5-9. Les partitions disjointes des classes du premier niveau des ontologies CMT (à gauche) et CRS-DR (à droite)

Pour chercher les mappings candidats à la fusion des deux ontologies CMT et CRS-DR 14*36=504 comparaisons sont nécessaires. Mais l'utilisation des 27 axiomes de disjonction entre classes (la recherche sous partitions disjointes) rend l'espace de recherche beaucoup plus réduit et seulement 155 comparaisons sont requises (par une analyse manuelle) pour détecter tous les mappings candidats à la fusion.

Il est évident que le temps d'exécution dépend aussi de la vitesse du processeur, du système d'exploitation ainsi que de la mémoire utilisée par le système. Mais notre objectif est seulement de monter que la recherche sous partitions disjointes réduit considérablement le nombre de comparaisons pour chercher les mappings candidats à la fusion ce qui réduit la complexité du temps d'exécution.

En revanche, pour garantir de ne pas rater des mappings corrects, si notre algorithme ne trouve pas des classes similaires dans les partitions disjointes, il cherche ensuite dans la hiérarchie de niveau supérieure et ainsi de suite jusqu'au balayage de toute la hiérarchie de l'ontologie. Ceci correspond à l'analyse exhaustive des ontologies sources mais en donnant la priorité à la recherche dans les partitions disjointes.

3. Partie II : Evaluation du système AOM-FOM

3.1. Evaluation des résultats obtenus par le système AOM-FOM

Dans cette section, nous allons évaluer les résultats obtenus par notre système pour aligner différentes ontologies.

3.1.1. Description des experimentations menées

Pour prouver que notre algorithme peut traiter des ensembles de données divers, avec un nombre d'éléments, de plus en plus croissant, et pour permettre la comparaison avec d'autres systèmes, nous avons procédé à des expériences d'alignement sur un benchmark standard et récent, *OAEI'2015*.

- **OAEI** est une initiative internationale coordonnée. Elle organise les évaluations d'un très grand nombre de systèmes d'alignement d'ontologies. L'objectif principal d'*OAEI* est de comparer les performances des systèmes d'alignement d'ontologies sur des cas de test précisément définis. *OAEI'2015* offre 8 tracks, tels que *Benchmarck*, *conference*, *interactive matching*, etc., avec 15 cas de test.

Alors, nous avons choisi d'aligner et/ou de fusionner quelques ontologies du track *biblio* (sous track du track *Benchmark*), en plus d'autres ontologies du track *conference*. Le domaine du track *biblio* est l'organisation et la gestion des références bibliographiques. Les ontologies du track *conference* sont élaborées par des chercheurs du domaine des conférences pour servir les systèmes de gestion et d'organisation des conférences. Et, elles sont utilisées par plusieurs chercheurs surtout pour tester les algorithmes et outils d'alignement d'ontologies (Ontology Matching). Dans ces expériences, nous avons choisi d'appliquer la distance d'édition pour mesurer la similarité lexicale et un seuil critique égal à un (threshold = 1). Nous avons calculé les mesures d'évaluation standard (Précision, Rappel et F-measure) contre les alignements de référence (referent alignments), fournis par la campagne OAEI'2015.

3.1.2. Analyse et discussion

Après la détection de mappings candidats à la fusion par un système de fusion d'ontologies (automatique ou semi-automatique), ces mappings sont comparés avec

ceux détectés manuellement par un expert humain, pour mesurer la précision des résultats obtenus.

Plusieurs mesures d'évaluation des résultats des mappings obtenus existent dans la littérature, en l'occurrence, la précision, le rappel, F-measure et over-all. Ces mesures sont calculées à base de quatre valeurs différentes mesurant les différents types ou formes de mappings détectés :

1. TP (True Positive) : Le nombre de mappings corrects : Signifiant que la classe c_a est correctement alignée avec la classe c_b, et ce mapping est également détecté par l'expert humain.
2. TN (True Negative) : Le nombre de dis-similarité correctes (correct not mappings) : Signifiant que la classe c_a est différente de la classe c_b et ceci correspond également à l'analyse de l'expert humain.
3. FP (False Positive) : Le nombre de mappings incorrects : Signifiant que la classe c_a est incorrectement alignée avec la classe c_b et ceci contredit l'analyse de l'expert humain.
4. FN (False Negative) : Le nombre de mappings ratés ou oubliés. Signifiant que la classe c_a n'est pas alignée avec la classe c_b, et selon l'expert humain les deux classes sont similaires.

   Alors, à base de ces quatre cas qui peuvent être détectés par le module de découverte de mappings candidats à la fusion, on peut calculer les mesures suivantes :

$$precision = \frac{TP}{TP + FP} \quad (1)$$

$$recall = \frac{TP}{TP + FN} \quad (2)$$

$$F - measure = \frac{2 * precision * recall}{precision + recall} \quad (3)$$

$$Over - All = \mathrm{Re}\, call * 2 - \frac{1}{precision} \quad (4)$$

3.1.3. Résultats expérimentaux

Comme déjà noté, à partir du track *conference*, nous avons choisi les ontologies *iasted*, *ekaw* et *cmt*, et à partir du track *biblio*, nous avons choisi les ontologies *101*, *201* et *202*.

Alors, pour mesurer les performances de notre système et les comparer avec ceux d'autres systèmes, nous avons utilisé quatre cas différents de test : (iasted Vs ekaw), (ekaw Vs cmt), (*101* Vs *201*) et (*101* Vs *202*). Les résultats expérimentaux obtenus par notre système sont d'abord comparés à ceux obtenus par PROMPT. Ensuite, Les performances de notre système sont comparées à celles des autres systèmes qui ont obtenu de bons résultats dans le benchmark *OAEI'2015*. A titre illustratif, nous avons choisi les systèmes suivant : *YAM++*, *DKP-AOM*, *Log-Map*, et *XMap++*. Les résultats expérimentaux de ces systèmes sont extraits du site web *d'OAEI'2015*. Pour consulter les résultats expérimentaux d'autres systèmes, le lecteur peut se référer à http://www.oaei.ontologymatching.org/2015/.

Cas de test 1: (*iasted* Vs *ekaw*)

Lors de la découverte de similarité entre les deux ontologies de conférence *iasted* et *ekaw*, notre système a détecté 7 mappings candidats à être fusionnés. Tous ces mappings sont corrects (i.e. *TP=7*) vu qu'ils sont tous détectés par l'expert humain (dans les mappings références), également. En plus des 5 mappings détectés par une simple analyse lexicale, notre système a également détecté le mapping entre les deux classes (*Session_Chair* Vs *Session_chair*), après l'application du mécanisme de stripping, permettant d'éliminer le caractère spécial tiret « _ », suivi par la normalisation, permettant de transformer tous les caractères du nom de la classe (*Session_Chair*) en minuscule et enfin, l'application de l'analyse lexicale. Un autre mapping détecté par notre système est le mapping entre les deux classes (*Paper_Author* Vs *Author*), après l'application du mécanisme de stripping suivi par l'application de la première heuristique (du niveau structurel) instruisant que si la chaîne de caractères étiquetant une classe est incluse dans la chaîne de caractères étiquetant la deuxième classe et si les deux classes ont au moins deux parents (s'ils existent, sinon un seul parent) qui sont similaires alors les deux classes sont similaires, alors,

Author est inclus dans *Paper_Author*

Et *Paper_Author* is-a *Person* (il a un seul parent)

Et *Author* is-a *Speaker* is-a *Delegate* is-a *Person*

Donc *Paper_Author* a un seul parent (*Person*) qui est similaire à un parent de *Author*

Alors *Paper_Author* est similaire à *Author*.

Il est à noter également que notre système n'a détecté aucun mapping erroné (qui n'a pas été détecté dans les mappings références (i.e. *FP=0*).

Alors, la précision atteinte par notre système est égale à 1 (i.e. *précision = 7/7+0=1*). Cependant, notre système n'a pas pu détecter les mappings suivants qui ont été détectés dans les mappings références: (*Conference_banquet* Vs *Dinner_Banquet*), (*Event* Vs *Activity*), (*Location* Vs *Place*).

Trois mappings sont ratés, alors (*FN=3*), et le Rappel obtenu par notre système est égale à 0.7 (i.e. *Rappel=7/7+3=0.7*) et la mesure F-measure est égale à 0.824 (i.e. *F-measure=2*1*0.7/1+0.7=0.824*) et le *over-all=0.7*.

Ce cas de test montre l'utilité du module de prétraitement pour améliorer les résultats du module basé sur l'analyse linguistique (au niveau schéma) lors de la comparaison des noms avec des conventions de nommage différentes, tel que l'utilisation des soulignements, des traits, des majuscules, etc..

La précision, le rappel, le F-measure et le over-all obtenus lors de la découverte de similarité entre les deux ontologies sont illustrés par la figure 5.10 :

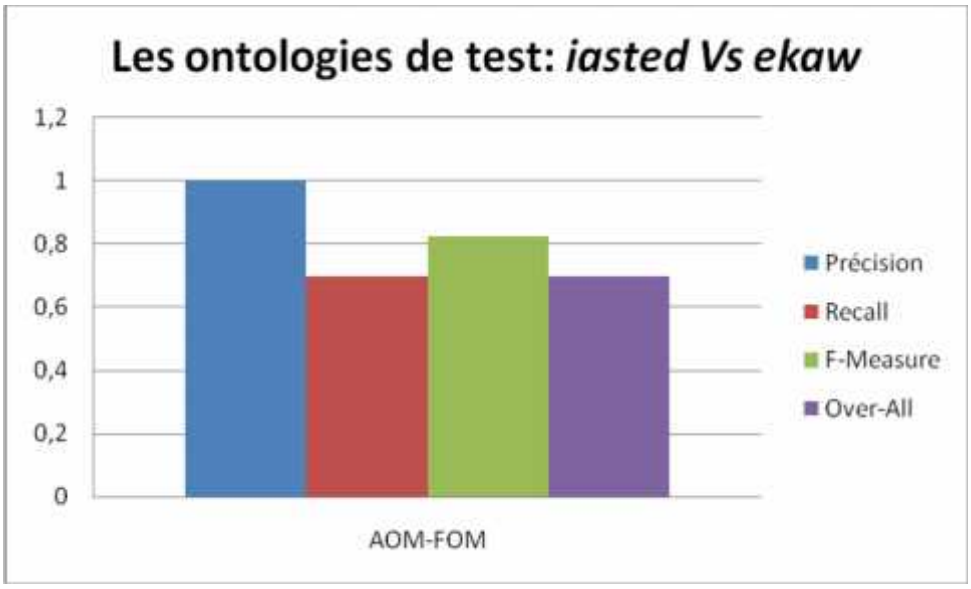


Figure 5-10. Les performances obtenues par AOM-FOM lors de l'alignement des ontologies iasted Vs ekaw

En comparant les résultats de notre système avec ceux obtenus par Prompt, nous pouvons voir que notre système a détecté des mappings plus précis que ceux détectés par Prompt, voir figure 5.11. En effet, alors que notre système a détecté 7 mappings et en a raté 3 (avec précision = 0.875 et rappel = 0.7), prompt a détecté uniquement 5 mappings et en a raté 5 (avec précision= 1 et rappel=5/5+5 = 0.5), F-measure=0.667 et over-all=0.667.

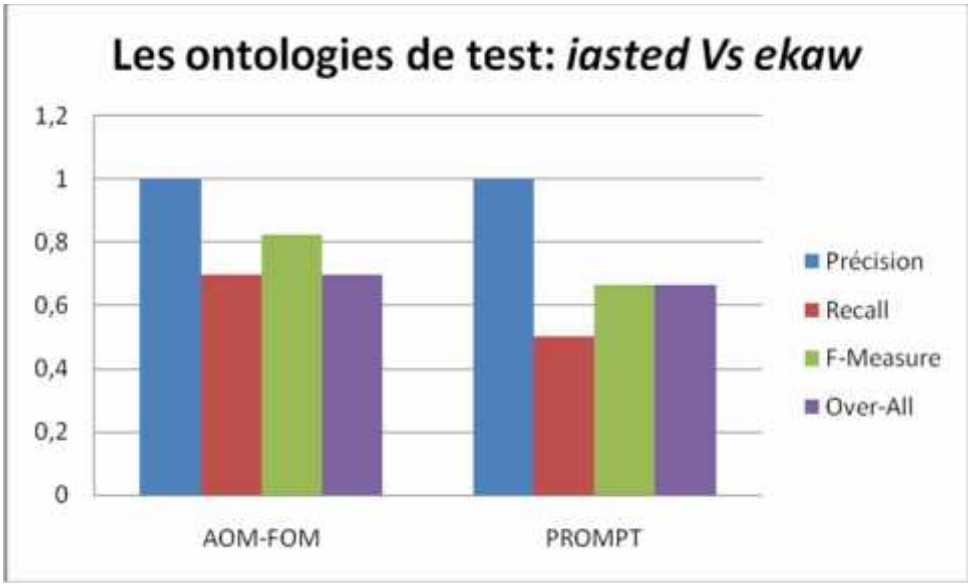


Figure 5-11. Comparaison des performances obtenues par AOM-FOM contre celles obtenues par Prompt, lors de l'alignement des ontologies iasted Vs ekaw

Maintenant, les résultats expérimentaux obtenus par notre système vont être comparés avec ceux obtenus par les systèmes (*YAM++, DKP-AOM, Log-Map, et XMap++*) et publiés dans le site web *d'OAEI'2015*. Une comparaison analytique est donnée par le tableau 5.1 et une comparaison graphique est illustrée par la figure 5.12.

Nous remarquons que notre système enregistre des performances similaires à celles obtenues par XMAP++ et meilleures que celles obtenues par les autres systèmes.

Table 5.1. Comparaison analytiques des performances obtenues par AOM-FOM contre celles obtenues par YAM++, DKP-AOM, Log-Map et XMap++.

	Precision	Recall	F-Measure	over-All
AOM-FOM	1	0,7	0,824	0,7
YAM++	0,778	0,7	0,737	0,5
DKP-AOM	1	0,6	0,75	0,6
LogMap	0,7	0,7	0,7	0,4
Xmap++	1	0,7	0,824	0,7

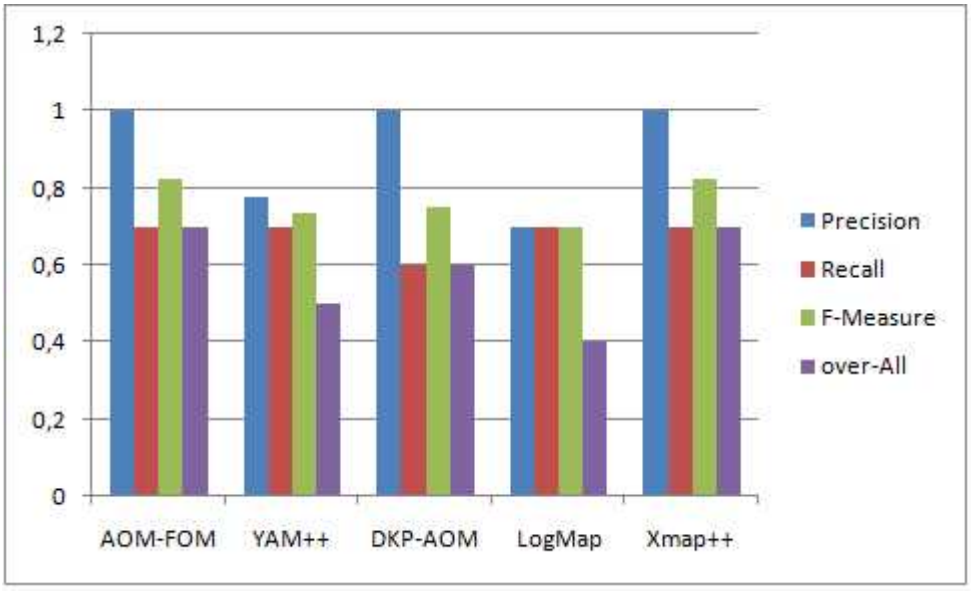


Figure 5-12. Comparaison graphique des performances obtenues par AOM-FOM contre celles obtenues par YAM++, DKP-AOM, Log-Map et XMap++.

Cas de test 2: (*ekaw* Vs *cmt*)

Lors de la découverte de similarité entre les deux ontologies de conférence *ekaw* et *cmt*, notre système a détecté 10 mappings candidats à la fusion. 9 de ces mappings sont corrects (i.e. TP=9) vu qu'ils sont tous détectés par l'expert humain (dans les mappings références), également. En plus des 5 mappings détectés par une simple analyse lexicale, notre système a également détecté le mapping entre les deux classes (*Author* Vs *Paper_Author*), après l'application du mécanisme de stripping suivi par l'application de la première heuristique (du niveau structurel) instruisant que si la chaîne de caractères étiquetant une classe est incluse dans la chaîne de caractères étiquetant la deuxième classe et si les deux classes ont au moins deux parents (s'ils existent, sinon un seul parent) qui sont similaires alors les deux classes sont similaires, donc,

Author est inclus dans *Paper_Author*

Et *Paper_Author* is-a *Person* (il a un seul parent)

Et *Author* is-a *User* is-a *Person*

Donc *Paper_Author* a un seul parent (*Person*) qui est similaire à un parent de *Author*

Alors *Paper_Author* est similaire à *Author*.

En plus de ces 6 (5+1) mappings corrects, notre système a également détecté trois autres mappings entre les propriétés d'objet suivantes :

AssignedTo Vs *hasReviewer*

hasBeenAssigned Vs *reviewerOfPaper*

writtenby Vs *reviewWrittenBy*,

en propageant la similarité détectée (entre les classes) par le module d'alignement au niveau entité à travers leurs propriétés d'objet par le module d'alignement au niveau structurel, et en utilisant l'heuristique de recherche de similarité qui instruit que 2 propriétés d'objet sont similaires si leurs domaines sont similaires et leurs co-domaines (ranges) sont également similaires. En effet, *AssignedTo* (*Paper→Reviewer*) est similaire à *hasReviewer* (*Paper→Possible_Reviewer*) car leurs domaines sont similaires (*Paper* Vs *Paper* , détecté par une simple analyse lexicale) et leurs co-domaines sont aussi similaires (*Reviewer→Possible_Reviewer*, détecté comme décrits ci-dessous, stripping+heuristique de contenu).

En plus, *writtenBy (Review→Reviewer*) est similaire à *reviewWrittenBy* (Review→*Reviewer*) car leurs domaines sont similaires (*Review* Vs *Review*, détecté par une simple analyse lexicale) et leurs co-domaines sont aussi similaires (*Reviewer* Vs *Reviewer*, détecté aussi, par une simple analyse lexicale).

Alors, en tout, nous avons 5+1+3=9 mappings corrects (i.e. *TP=9*).

Notre système a également détecté le mapping entre les deux classes (*Reviewer* Vs *Possible_Reviewer*), après l'application du mécanisme de stripping suivi par l'application de la première heuristique (du niveau structurel) instruisant que si la chaîne de caractères étiquetant une classe est incluse dans la chaîne de caractères étiquetant la deuxième classe et si les deux classes ont au moins deux parents (s'ils existent, sinon un seul parent) qui sont similaires alors les deux classes sont similaires, donc,

Reviewer est inclus dans *Possible_Reviewer*

Et *Possible_Reviewer* is-a *Person* (il a un seul parent)

Et *Reviewer* is-a *User* is-a *Person*

Donc *Possible_Reviewer* a un seul parent (*Person*) qui est similaire à un parent de *Reviewer* Alors *Possible_Reviewer* est similaire à *Reviewer.*

Cependant, ce mapping est erroné car il n'est pas détecté dans les mappings de références et donc (*FP=1*).

D'un autre côté, notre système a raté 2 mappings (*ConferenceMember* Vs *ConferenceParticipant*) et (*PaperFullVersion* Vs *RegularPaper*) qui ont été détectés dans les mappings références et donc (*FN=2*).

En résumé, (*TP=9, FP=1, FN=2*) alors, (*Précision =0.9, Rappel=0.818, F-measure=0.857 et le over-all=0.727*).

Ce cas de test montre le potentiel du module de détection de similarité au niveau structurel, pour découvrir plus de mappings.

La précision, le rappel, le F-measure et le over-all obtenus lors de la découverte de similarité entre les deux ontologies sont illustrés par la figure 5.13:

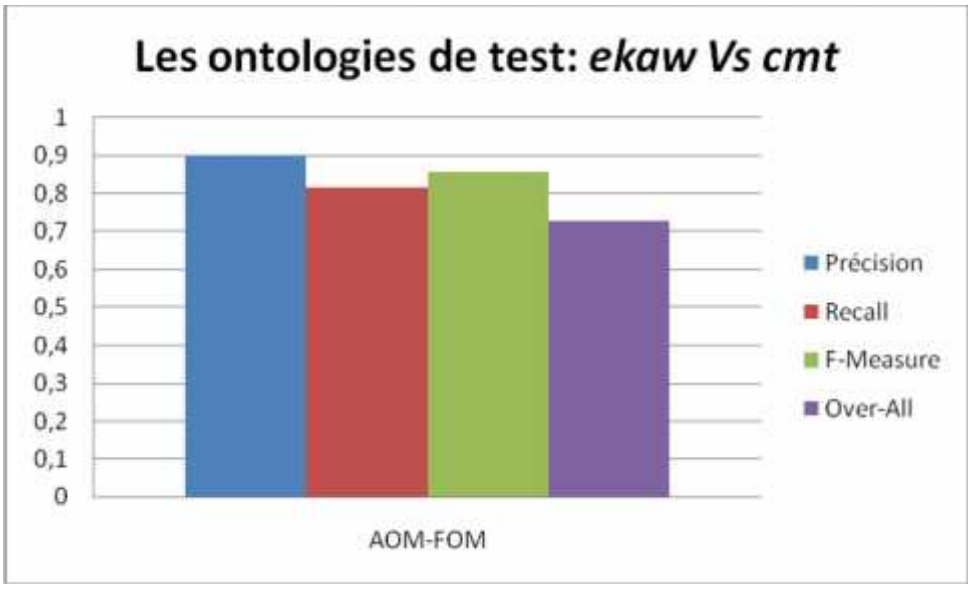


Figure 5-13. Les performances obtenues par AOM-FOM lors de l'alignement des ontologies ekaw Vs cmt

En comparant les résultats de notre système avec ceux obtenu par Prompt, nous constatons que notre système a détecté des mappings plus précis que ceux détectés par Prompt, voir figure 5.14. En effet, alors que notre système a détecté 9 mappings et en a raté 2 (avec précision = 0.9 et rappel = 0.818), prompt a détecté uniquement 5 mappings et en a raté 6 (avec précision= 1 et rappel=5/5+6 = 0.454, F-measure=0.624 et over-all=0.454).

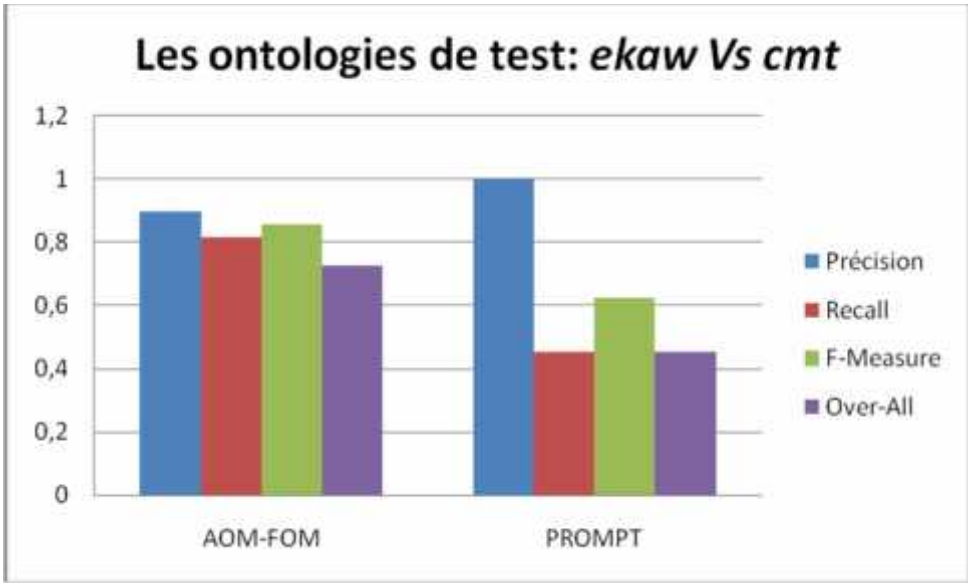


Figure 5-14. Comparaison des performances obtenues par AOM-FOM contre celles obtenues par Prompt, lors de l'alignement des ontologies ekaw Vs cmt.

Maintenant, les résultats expérimentaux obtenus par notre système vont être comparés par ceux obtenus par les systèmes (*YAM++, DKP-AOM, Log-Map, et XMap++*) et publiés dans le site web *d'OAEI'2015*. Une comparaison analytique est donnée par le tableau 5.2 et une comparaison graphique est illustrée par la figure 5.15.

Nous remarquons que les précisions obtenues par les systèmes DKP-AOM et XMap++ sont meilleures que celles obtenues par notre système (pas de mappings erronés). Mais le rappel obtenu par notre système est meilleur que celui obtenu par eux (ils ont raté plus de mappings corrects). En plus, notre système enregistre des performances meilleures à celles obtenues par YAM++ et LogMap.

Table 5.2. Comparaison analytique des performances obtenues par AOM-FOM contre celles obtenues par YAM++, DKP-AOM, Log-Map et XMap++.

	Precision	Recall	F-Measure	over-All
AOM-FOM	0,9	0,818	0,857	0,727
YAM++	0,75	0,545	0,632	0,363
DKP-AOM	1	0,455	0,625	0,455
LogMap	0,75	0,545	0,632	0,363
Xmap++	1	0,545	0,706	0,545

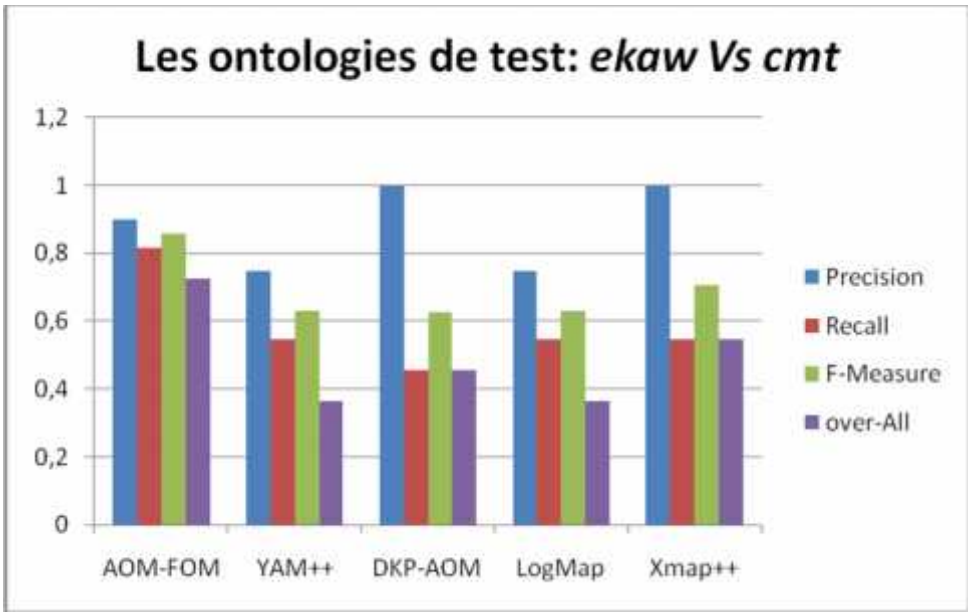


Figure 5-15. Comparaison graphique des performances obtenues par AOM-FOM contre celles obtenues par YAM++, DKP-AOM, Log-Map et XMap++.

Cas de test 3 (*101 Vs 201*)

Lors de la découverte de similarité entre les deux ontologies *biblio 101* et *201*, notre système a détecté 93 mappings candidats à la fusion. Tous ces mappings sont corrects (*i.e. TP=93*) vu qu'ils sont tous détectés dans les mappings références, également. En plus des 89 mappings détectés par une simple analyse lexicale des descriptions (ou de commentaires) des classes et de leurs propriétés et/ou de classification de leurs instances, et un seul mapping détecté par une simple analyse lexicale des noms de propriétés (*lastName* Vs *lastName*), notre système a également détecté les mappings suivants entre les propriétés de type de données, en appliquant l'heuristique instruisant que si deux propriétés partagent le même domaine et le même co-domaine alors elles sont similaires, et vu que les trois propriétés :

(day : gDay), (month : gMonth) et (year : gYear)

ainsi que les trois propriétés :

(dznadzh : gDay), (asndzsd : gMonth) et (zdsnzsdn : gYear)

ont des types (co-domaines) différents dans leurs domaines respectifs, alors, cette heuristique est applicable.

Il est à noter également que notre système n'a pas détecté des mappings erronés (qui n'ont pas été détecté dans les mappings références (*i.e.FP=0*).

Alors, la précision atteinte par notre système est égale à 1 (i.e. ***precision*** = *93/93+0=**1***). Cependant, notre système n'a pas pu détecter quatre mappings, qui ont été détectés dans les mappings références, qui sont:

(*city* Vs *zdzndh*), (*name* Vs *dszabd*), (*shortName* Vs *dsza*) et (*state* Vs *zdnzadh*).

Alors (***FN=4***), et donc :

Rappel**=93/93+4=**0.959

F-measure**=2*1*0.959/1+0.959=**0.979

***over-all**=0.959*(2-1/1)=**0.959**.*

La précision, le rappel, le F-measure et le over-all obtenus lors de la découverte de similarité entre les deux ontologies sont illustrés par la figure 5.16:

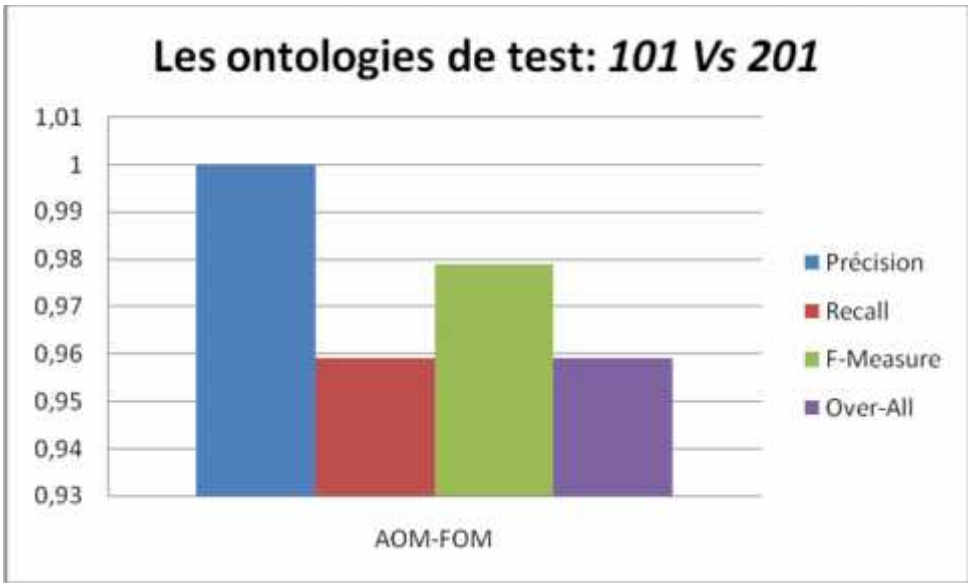


Figure 5-16. Les performances obtenues par AOM-FOM lors de l'alignement des ontologies 101 Vs 201

En comparant les résultats de notre système avec ceux obtenus par Prompt, nous constatons que notre système a détecté des mappings plus précis que ceux détectés par Prompt, voir figure 5.17. En effet, alors que notre système a détecté 93 mappings et en a raté 4 (avec précision = 1 et rappel = 0.959), prompt a détecté uniquement un seul mapping (*lastName* Vs *lastName*) et en a raté 92 mappings (avec précision= 1 et rappel=1/1+92 = 0.11), F-measure=0.198 et over-all=0.11.

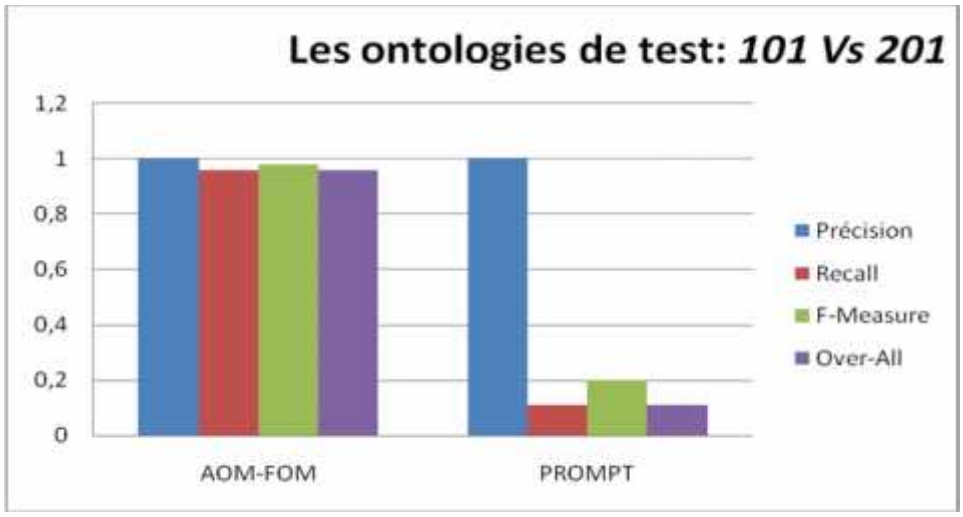


Figure 5-17. Comparaison des performances obtenues par AOM-FOM contre celles obtenues par Prompt, lors de l'alignement des ontologies 101 Vs 201

Maintenant, les résultats expérimentaux obtenus par notre système vont être comparés avec ceux obtenus par les systèmes (*YAM++, DKP-AOM, Log-Map, et XMap++*) et publiés dans le site web *d'OAEI'2015*. Une comparaison analytique est donnée par le tableau 5.3 et une comparaison graphique est illustrée par la figure 5.18.

Nous remarquons que notre système enregistre de meilleures performances que celles enregistrées par les autres systèmes.

Il est à noter que le système DKP-AOM n'a pas pu détecter aucun mappings (corrects ou erronés) car il ne compare que les noms de classes et des propriétés alors que tous ces noms sont remplacés par des chaînes de caractères aléatoires dans l'ontologie 201.

Table 5.3. Comparaison analytique des performances obtenues par AOM-FOM contre celles obtenues par YAM++, DKP-AOM, Log-Map et XMap++.

	Precision	Recall	F-Measure	over-All
AOM-FOM	1	0,959	0,979	0,959
YAM++	0,978	0,938	0,973	0,917
DKP-AOM	n/a	n/a	n/a	n/a
LogMap	1	0	0	0
Xmap++	1	0	0	0

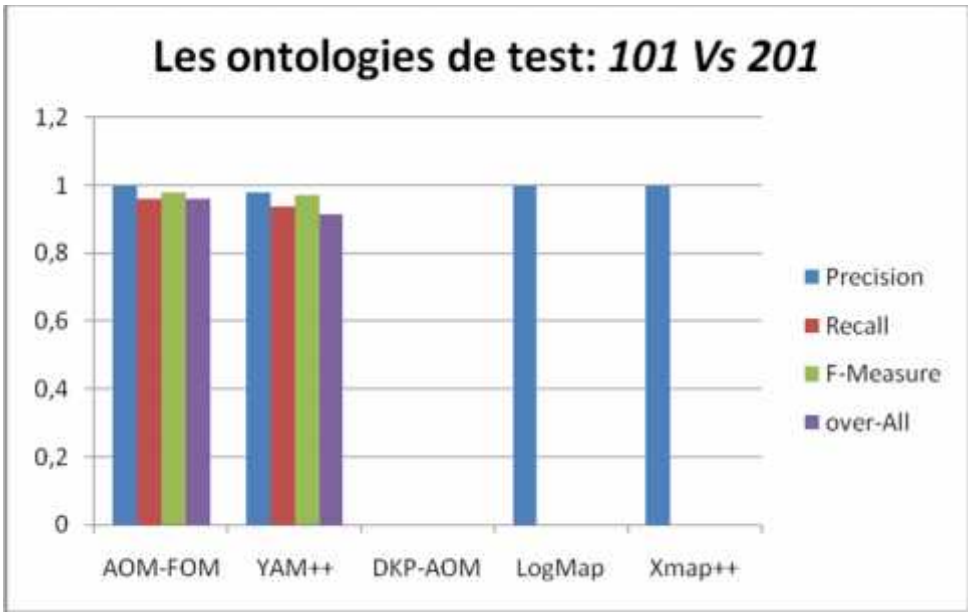


Figure 5-18. Comparaison graphique des performances obtenues par AOM-FOM contre celles obtenues par YAM++, DKP-AOM, Log-Map et XMap++.

Cas de test 4 (*101 Vs 202*)

Lors de la découverte de similarités entre les deux ontologies *biblio* (représentant les références bibliographiques) *101* et *202*, notre système a donné des résultats relativement bas car il n'y a aucune information linguistique dans l'ontologie cible (pas de noms, pas de commentaires, les noms de classes et de propriétés sont remplacés par des chaînes de caractères aléatoires et les commentaires sont supprimés) et seulement 13 classes sont équipées de leurs instances. Cependant, ce cas de test montre l'efficacité et l'utilité du module de détection de similarité basé sur la classification des instances (au niveau extensionnel) quand on n'a pas des informations linguistiques représentées dans les ontologies sources (le module de détection de similarités au niveau schéma ne rend pas de résultats). Alors, en plus des 13 mappings détectés par la classification automatique des instances de classes entre les classes *(article, conference, Date, InBook, InProceedings, Journal, Misc, Monograph, PageRange, PersonList, Proceedings, Publisher, Address)*, notre système a également détecté le mapping entre les deux propriétés *(LastName Vs LastName)* par l'application d'une simple analyse lexicale de leurs noms (exceptionnellement, *LastName* est le seul nom qui est resté dans les ontologies source). Après cela, 10 autres mappings entre les propriétés d'objets ont été détectés en appliquant l'heuristique (de propagation de similarité) instruisant que si deux propriétés d'objets

partagent le même domaine et le même co-domaine (range) alors elles sont similaires. Il s'agit donc des propriétés d'objets suivantes :

- ✓ ***address*** *(thing→Address),*
- ✓ ***articles*** *(journal→Article),*
- ✓ ***communications*** *(Proceedings → InProceedings),*
- ✓ ***date*** *(reference → Date)*
- ✓ ***event*** *(Proceedings → conference),*
- ✓ ***book*** *(InBook→ Monograph),*
- ✓ ***journal*** *(Article → Journal),*
- ✓ ***Proceedings*** *(InProceedings → Proceedings),*
- ✓ ***location*** *(Reference → Address),*
- ✓ ***publisher*** *(Reference → publisher).*

Et en plus de ces mappings, 6 autres mappings entre les propriétés de type de données ont été détectés en appliquant la même heuristique mais sur les propriétés de type de données qui n'ont pas le même type (co-domaine) dans la même classe qu'ils décrivent (le domaine). Ces mappings sont *Day (Date → gDay), Month (Date → gMonth), Year (Date → gYear), issue (NumberOrVolume → string), howPublished (Misc → string), periodicity (Journal → string).*

En résumant, notre système a pu détecter 30 mappings candidats à la fusion. Tous ces mappings sont corrects (i.e. *TP =30*) vu qu'ils sont tous détectés dans les mappings références.

Il est à noter également, que notre système n'a pas détecté de mappings erronés (i.e. FP=0), alors, la précision atteinte par notre système est égale à 1 (*i.e.* ***precision=****30/30+0=****1***). Cependant, notre système n'a pas pu détecter 67 mappings corrects qui ont été détectés dans les mappings références (*i.e. FN = 67*), ceci est dû au fait qu'il n'existe aucune information linguistique décrivant les classes qui correspondent (pas de noms, pas de commentaires). De plus, ces classes (les classes qui correspondent aux mappings ratés) n'ont pas d'instances représentées dans l'ontologie cible (202). Alors :

recall=*30/30+67=****0.309***

F-measure =*2*1*0.309/1+0.309=****0.472***

***over-All**=0.309*(2-1/1)=**0.309**.*

La précision, le rappel, la F-measure et le over-All obtenus lors de la découverte de similarités entre ces deux ontologies sont illustrés par la figure 5.19:

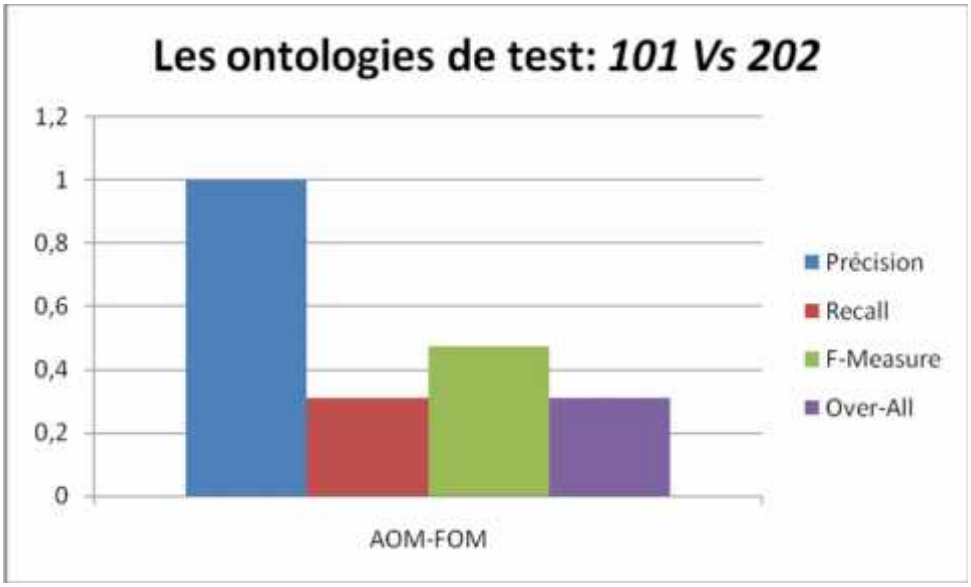


Figure 5-19. Les performances obtenues par AOM-FOM lors de l'alignement des ontologies 101 Vs 202

En comparant les résultats de notre système avec ceux obtenu par Prompt, nous constatons que notre système a détecté des mappings plus précis que ceux détectés par Prompt, voir figure 5.20. En effet, alors que notre système a détecté 30 mappings et en a raté 67 (avec précision = 1 et rappel = 0.309), prompt a détecté uniquement un seul mapping (*lastName* Vs *lastName*) et en a raté 92 (avec précision= 1 et rappel=1/1+92 = 0.11), F-measure=0.198 et over-all=0.11.

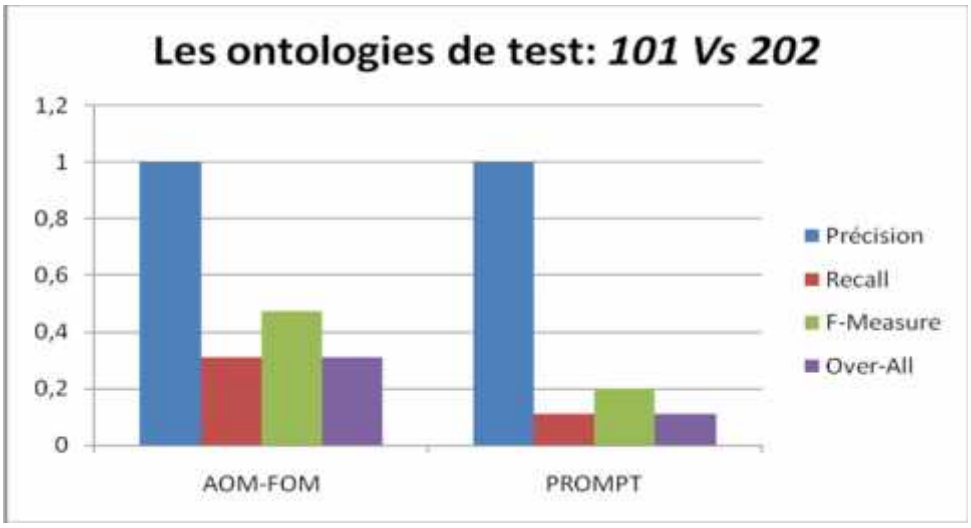


Figure 5-20. Comparaison des performances obtenues par AOM-FOM contre celles obtenues par Prompt, lors de l'alignement des ontologies 101 Vs 202

Maintenant, les résultats expérimentaux obtenus par notre système vont être comparés avec ceux obtenus par les systèmes (*YAM++, DKP-AOM, Log-Map, et XMap++)* et publiés dans le site web *d'OAEI'2015*. Une comparaison analytique est donnée par le tableau 5.4 et une comparaison graphique est illustrée par la figure 5.21.

Nous remarquons que notre système enregistre de meilleures performances que celles enregistrées par les autres systèmes.

Il est à noter que le système DKP-AOM n'a pas pu détecter aucun mappings (corrects ou erronés) car il ne compare que les noms des classes et des propriétés alors qu'ils sont remplacés par des chaînes de caractères aléatoires dans l'ontologie 202.

Table 5.4. Comparaison analytique des performances obtenues par AOM-FOM contre celles obtenues par YAM++, DKP-AOM, Log-Map et XMap++.

	Precision	Recall	F-Measure	over-All
AOM-FOM	1	0,309	0,472	0,309
YAM++	1	0,278	0,435	0,278
DKP-AOM	n/a	n/a	n/a	n/a
LogMap	1	0	0	0
Xmap++	1	0	0	0

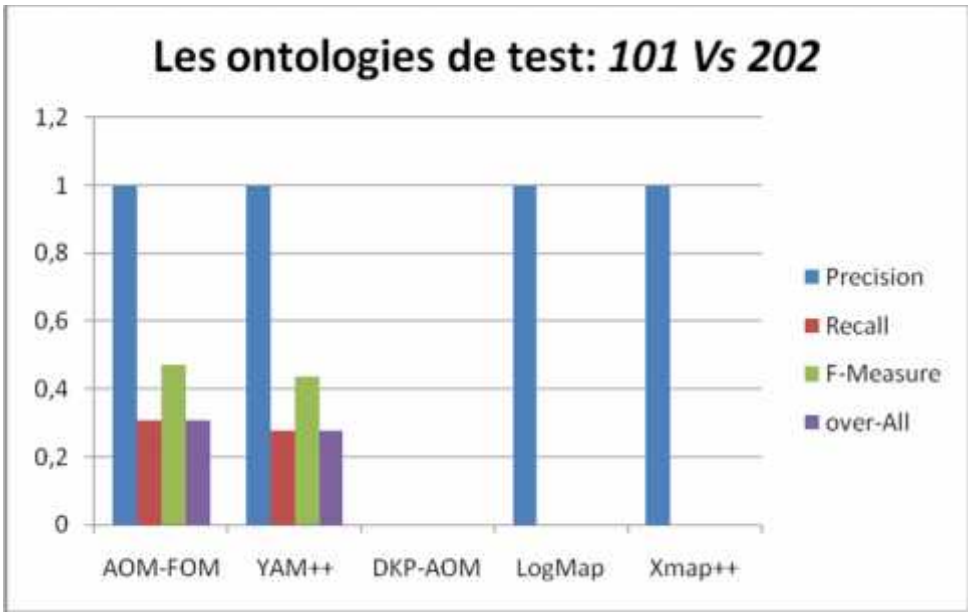


Figure 5-21. Comparaison graphique des performances obtenues par AOM-FOM contre celles obtenues par YAM++, DKP-AOM, Log-Map et XMap++.

- ***Conclusion*** : En comparant les résultats de notre algorithme avec ceux obtenus par d'autres systèmes, nous concluons que notre algorithme est compétitif avec les systèmes les mieux classés.

  Dans la plupart des cas, AOM-FOM donne une précision proche de 1, ce qui est très bon pour un processus de fusion (il évite la création d'incohérences dans l'ontologie fusionnée résultante). Cependant, parfois, il peut rater des correspondances (rappel inférieur à 1), ce qui crée des redondances dans l'ontologie résultat de la fusion.

Dans la section suivante, nous allons présenter des exemples d'exécution de notre algorithme pour fusionner des ontologies OWL-DL décrites dans différents scénarios.

3.2. Fusion de diverses ontologies avec le framwork AOM-FOM

3.2.1. Fusion d'ontologies du domaine d'organisation de conférences

A titre illustratif, la figure 5.22 illustre l'ensemble des mappings obtenus lors de l'alignement des ontologies *ekaw* Vs *cmt* (cas de test 2), des portions des ontologies source sont apparues en haut de la figure et les mappings entre eux sont en bas.

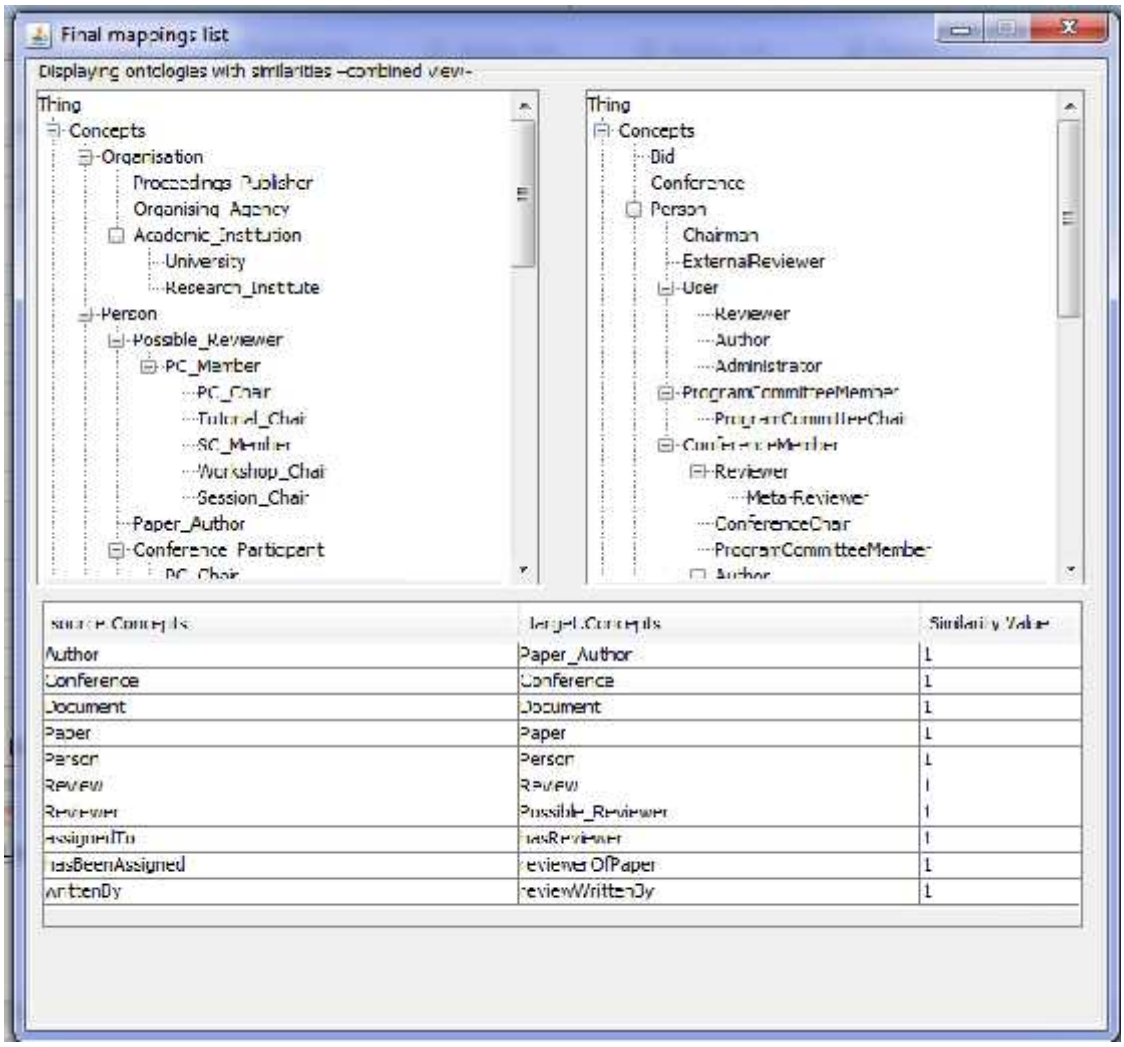


Figure 5-22. L'ensemble des mappings (en bas) obtenus lors de l'alignement des ontologies ekaw Vs cmt (en haut), (combined view).

L’ontologie résultat de la fusion de ces mappings est illustrée par la figure 5.23.

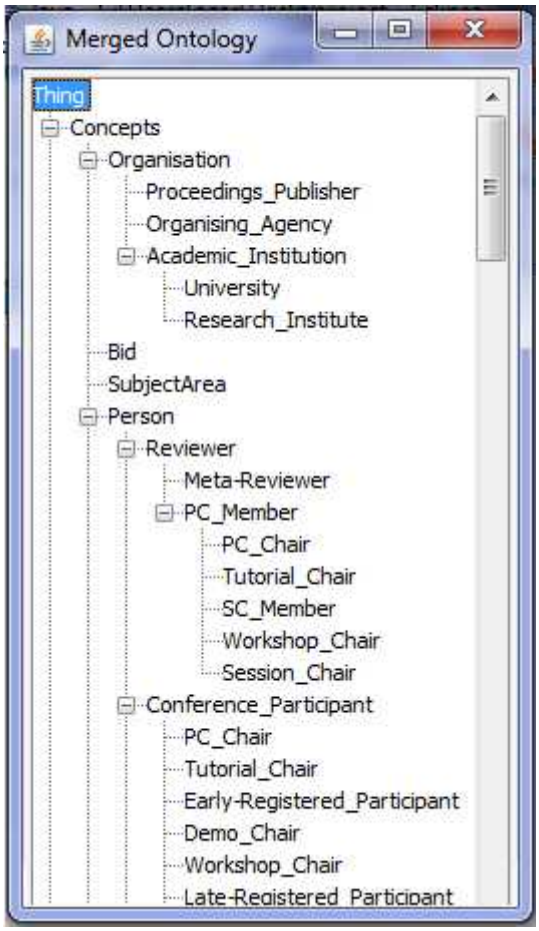


Figure 5-23. L'ontologie résultat de la fusion des deux ontologies ekaw Vs cmt.

3.2.2. Fusion d'ontologies du domaine industriel représentant les Turbines à vapeur

Les expérimentations effectuées sur les ontologies précédentes (les ontologies de conférence) ne permettent pas de démontrer le potentiel de tous les modules qui composent notre système. En d'autres termes, les expérimentations effectuées sur les ontologies de conférence ne démontrent pas le potentiel du module basé sur l'apprentissage automatique des instances de classes en utilisant les arbres de décision. C'est pourquoi nous avons choisi d'appliquer notre système pour fusionner des ontologies peuplées (des ontologies équipées par les instances de classes) du domaine industriel. Dans ce qui suit, nous allons appliquer notre système pour fusionner deux ontologies, *Topo* et *Diagno* qui représentent les turbines à vapeur utilisées pour la génération de l'électricité, mais selon deux différents contextes, topologique et diagnostique. Les deux ontologies sont automatiquement construites en utilisant l'outil

Data Master (un plugin sous protégé 2000), à partir de deux bases de connaissances sauvegardées sous l'éditeur Microsoft Office Access, CentraleTopo et Diagnostic, respectivement :

1. L'ontologie topo décrit les caractéristiques topologiques de la turbine à vapeur tout en décrivant ses composantes à travers des critères topologiques en l'occurrence : le code du composant, sa description, sa zone de localisation, sa famille, sa fonction ainsi que son code-parent.
2. L'ontologie diagno décrit les cas d'intervention sur les composants de la turbine à vapeur pour maintenance ou réparation. Elle ne décrit donc que les composants ayant subi des interventions techniques lors d'un intervalle de temps bien spécifié, où chaque cas d'intervention sur un ou plusieurs composants est décrit par : le code du composant, sa description, le code de son parent géographique, le code du symptôme, la description du symptôme, le code de défaut, la description de défaut, le code de la cause, la description de la cause, le code du remède et la description du remède,

   En haut à gauche de la figure 5.24 est illustré un schéma graphique de l'ontologie Topo, alors que la partie à droite représente l'ontologie Diagno.

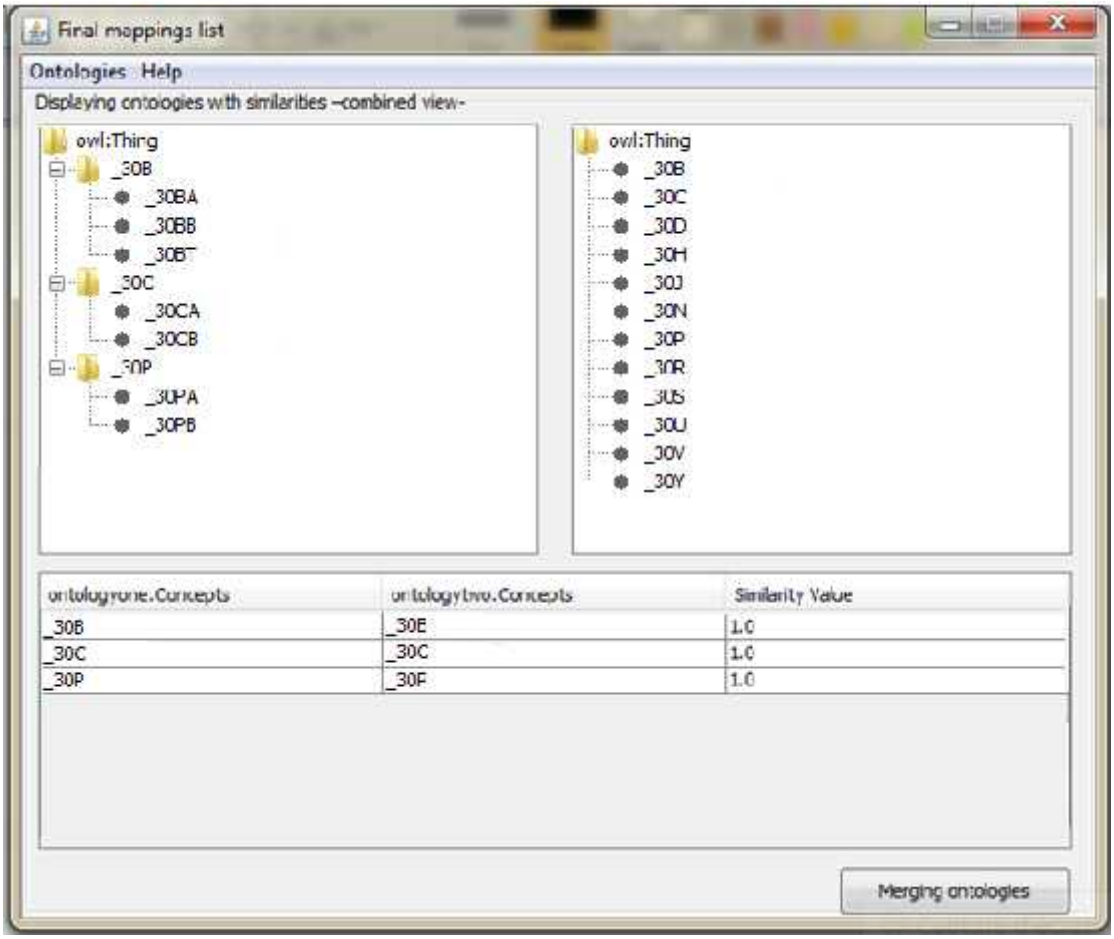


Figure 5-24. Les ontologies sources diagno (à droite) et topo (à gauche), combinées avec leurs classes similaires, combined view.

En se basant sur les classes similaires détectées par notre algorithme (voir la partie basse de la figure 5.24), l'ontologie résultat de la fusion par AOM-FOM est illustrée par la figure 5.25.

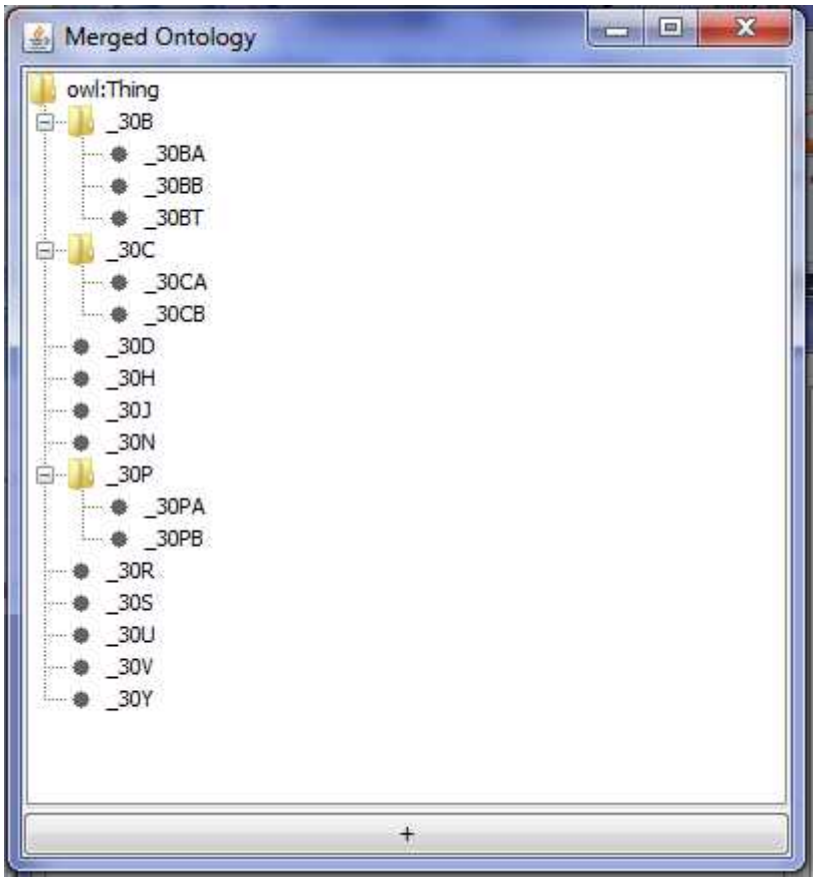


Figure 5-25. Ontologie résultat de la fusion

3.2.3. Fusion d'ontologies représentant le département d'informatique

Dans ce qui suit, nous allons présenter un exemple d'expérimentation de notre système pour fusionner des ontologies représentant le département d'informatique. Nous avons choisi de fusionner les deux ontologies présentées dans la partie haute de la figure 5.26. Alors, comme c'est déjà présenté par la partie basse de la figure 5.26, notre système a détecté 8 mappings entre les deux ontologies sources. En plus des 6 mappings détectés par une simple analyse lexicale des noms de classes, notre système a également détecté les deux mappings entre les classes (*prof Vs professor*) et (*impermanent_staff Vs temporary_staff*) en détectant leurs sens dans le dictionnaire WordNet.

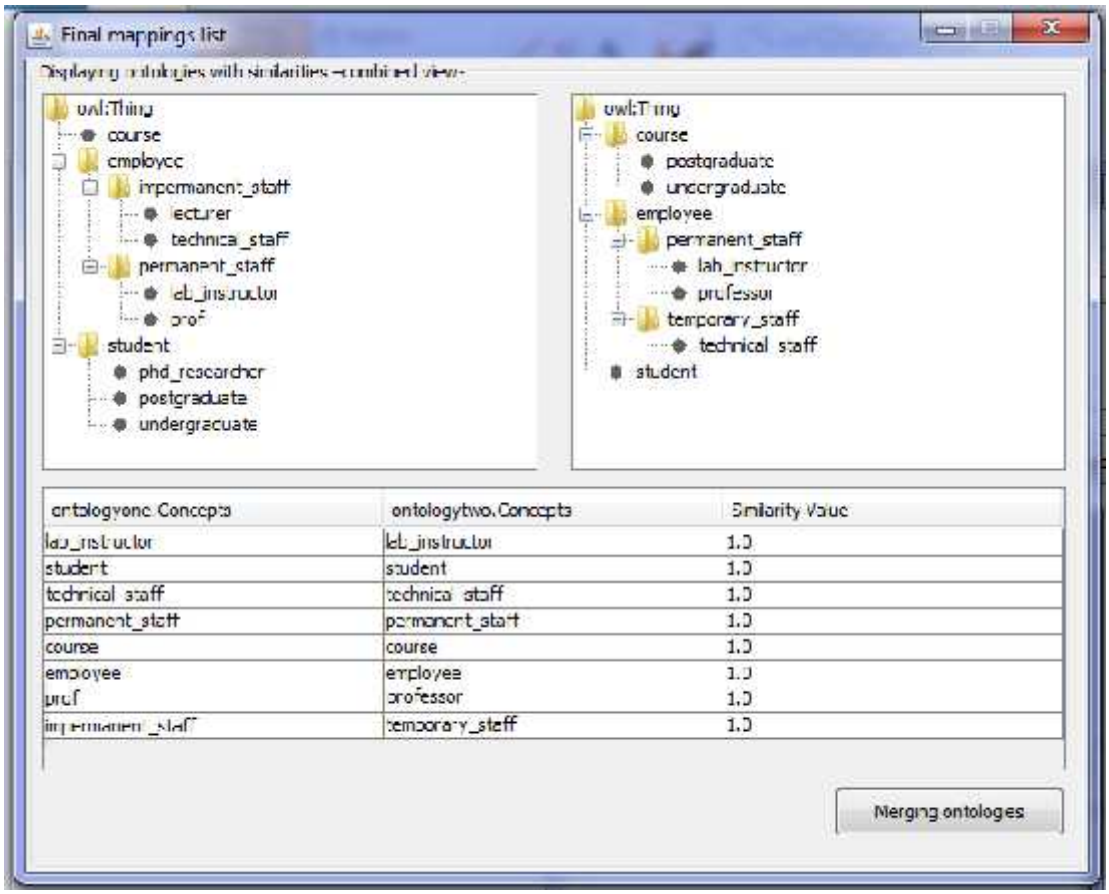


Figure 5-26. Les ontologies sources avec leurs similarités (mappings), combined view.

En effet, notre système a initialement détecté les mappings entre les classes (*undergraduate* Vs *undergraduate) et (postgraduate* Vs *postgraduate).*

Dans la première ontologie *undergraduate* est un étudiant (*student*) universitaire qui n'a pas encore reçu le premier degré (licence) alors que dans la deuxième ontologie *undergraduate* est relié aux études du premier cycle (*undergraduate courses).*

En plus, Dans la première ontologie *postgraduate* est un étudiant (*student*) universitaire qui poursuit ses études de post-graduation (après l'obtention du diplôme de licence) alors que dans la deuxième ontologie *postgraduate* est relatif aux études de la post-graduation (*postgraduate courses*).

- Ces hétérogénéités de conception résultent de l'utilisation de la même terminologie dans les deux ontologies pour modéliser des interprétations différentes.
- Ces mappings erronés, s'ils persistent ils produisent une ontologie fusionnée erronée et incohérente. Heureusement, ils (les deux mappings erronés) sont détectés comme étant des conflits de subsomption/disjonction et éliminés de la liste finale des

mappings valides, par le module de validation intégré dans notre système (voir le chapitre précédent).

En effet, étant donné le mapping initial : (*student* Vs *student*), et *undergraduate* (respectivement, *postgraduate*) de la première ontologie est initialement détecté comme étant similaire à la classe *undergraduate* (respectivement, *postgraduate*) de la deuxième ontologie.

Cependant, *student* est une super-classe (un père) de *undergraduate* (respectivement, *postgraduate*) dans la première ontologie et il (*student*) est disjoint avec *undergraduate* (respectivement, *postgraduate*) dans la deuxième ontologie.

Alors, en appliquant le deuxième modèle (pattern) (tel que présenté dans le chapitre précédent) par le module de validation intégré dans notre système, les deux mappings (*undergraduate* Vs *undergraduate) et (postgraduate* Vs *postgraduate)* sont détectés comme étant des mappings erronés (conflicting mappings) et sont éliminés de la liste finale des mappings valides, (voir la partie basse de la figure 5.26).

Finalement, notre système utilise la liste finale des mappings valides pour générer une ontologie fusionnée complète et cohérente, telle que présentée par la figure 5.27

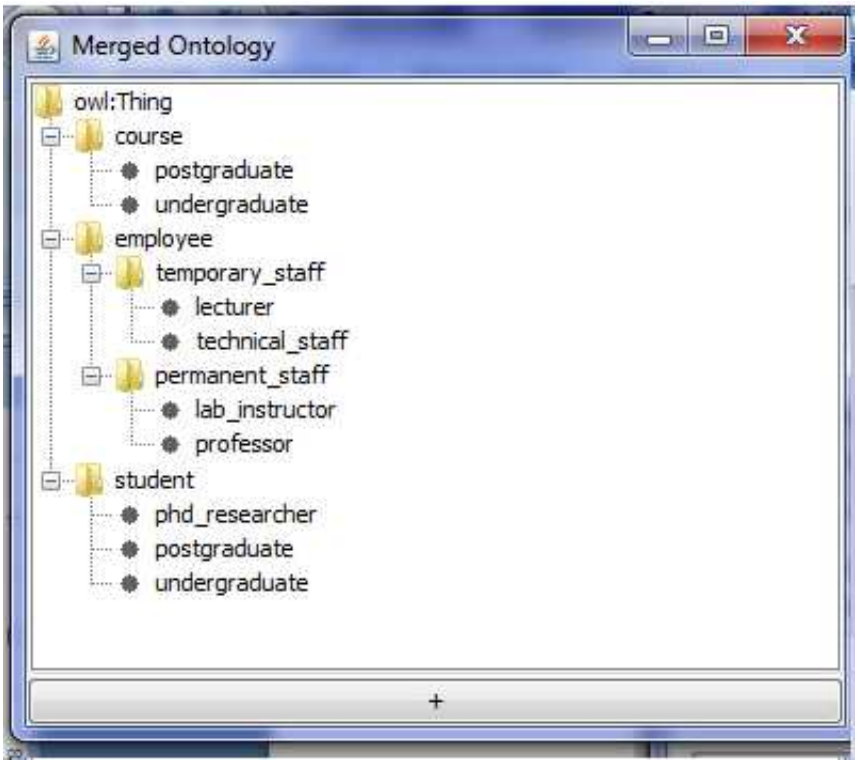


Figure 5-27. L'ontologie résultat de la fusion

4. Conclusion

Dans ce chapitre, nous avons présenté le prototype de notre système de fusion automatique d'ontologies, AOM-FOM. Nous avons montré que l'implémentation de notre prototype est principalement basée sur les APIs : *Jena*, *OWL-API*, *Sim-Metrics et MorphAdorner*. *Jena* et *OWL-API* extraient et sauvegardent les graphes d'ontologies dans la mémoire ce qui rend leurs exploration et manipulation facile et rapide. *Sim-Metrics API* fournit à l'utilisateur final une flexibilité de choix de l'algorithme de calcul de similarité lexicale selon le domaine d'application des ontologies sources. *MorphAdorner API* joue un rôle primordial dans le prétraitement des entités ontologiques en les ramenant à leurs formes de base. Un des algorithmes de prétraitement les plus important et celui qui permet de détecter les mappings entre les mots composés surtout lors du mapping des propriétés. Finalement, nous avons présenté plusieurs expérimentations de différents cas de test de notre système et comparé ses performances avec différents systèmes de fusion et /ou de mapping tels que PROMPT et ceux participant dans la campagne OAEI, 2015.

En résumé, nous pouvons affirmer que notre algorithme AOM-FOM est assez performant pour la fusion des ontologies hétérogènes, au vue des résultats expérimentaux obtenus.

Conclusions et perspectives :

Le Web sémantique est principalement fondé sur la notion d'ontologie comme étant un outil technologique clé pour assurer l'interopérabilité entre des applications hétérogènes et construire un environnement permettant aux utilisateurs humains et /ou machines de se communiquer et de se collaborer. Cependant, le nombre d'ontologies développées et maintenues sur le web est en progression absolue. Pour faire face à la nature dynamique des applications du web sémantique, en assurant une interopérabilité automatique entre les utilisateurs humains et / ou machines, la fusion d'ontologies est une solution incontournable.

La thématique de recherche développée dans ce travail de thèse s'inscrit dans le domaine de la fusion d'ontologies hétérogènes. Plus précisément, nous nous sommes intéressée à la conception et au développement d'un nouvel algorithme de fusion automatique d'ontologies, AOM-FOM. Il s'agit d'un outil de fusion automatique d'ontologies qui utilise un algorithme flexible d'alignement. Nous avons basé notre algorithme sur la combinaison de plusieurs modules d'alignement pour détecter les mappings candidats à la fusion. En l'occurrence, le module d'alignement au niveau entité et le module d'alignement au niveau structure. Le module d'alignement au niveau entité lui-même est construit de la combinaison du module d'alignement au niveau schéma et le module d'alignement au niveau extensionnel. Le module d'alignement au niveau schéma est basé sur le calcul de la similarité lexicale et / ou sémantique entre les chaînes de caractères étiquetant les classes ainsi que leurs propriétés (Propriétés de Type de Données, PTD et Propriétés d'Objet, PO) pour extraire l'ensemble de mappings candidats à la fusion entre les classes ainsi que leurs propriétés des ontologies sources. Ces mappings sont enrichis par les mappings du niveau extensionnel qui sont détectés en utilisant la technique de classification basée sur les arbres de décision pour classifier les instances des classes de l'ontologie cible avec leurs instances similaires de l'ontologie source sous la classe similaire appropriée. L'utilisation de la technique des arbres de décision ne distingue pas seulement notre algorithme de la plupart de ceux qui existent dans la littérature mais aussi, évite les limitations des méthodes linguistiques basées sur le Traitement de Langue Naturelle

car ils sont indépendants de tout type de dictionnaire et/ou de base de données lexicale. Les mappings obtenus au niveau entité sont également, enrichis par l'ensemble de mappings détectés au niveau structurel qui applique un ensemble de règles d'heuristiques pour diffuser les similarités précédemment obtenus à travers les structures des ontologies sources.

L'ensemble de mappings ainsi obtenus sont finalement vérifiés, raffinés et combinés par un module de validation qui utilise quelque pattern de détection de mappings incohérents, avant d'être utilisés pour générer l'ontologie fusionnée globale.

Nous avons basé notre algorithme également sur une stratégie de priorité qui donne la priorité à la recherche des mappings candidats à la fusion dans les partitions disjointes. Cette stratégie minimise, considérablement le nombre de comparaisons pour détecter les mappings candidats à la fusion, et donc augmenter les performances du système et réduire la consommation des ressources machine, i.e. le temps d'exécution et la mémoire de stockage.

A l'issue de ce travail, nous assumons que le domaine de la fusion d'ontologies hétérogènes, et surtout la fusion automatique d'ontologies a défini une base dure pour notre travail de recherche, mais il reste toujours une voie de recherche ouverte et très vaste, plusieurs points s'avèrent intéressants à explorer et à développer. Nous avons plus particulièrement identifié les points suivants qu'on estime particulièrement intéressants:

- D'abord, nous envisageons d'intégrer un module d'annotation automatique et sémantique des ontologies sources pour les enrichir par des données d'instances, à partir de documents des domaines d'application. Par conséquent, la stratégie basée-instance sera applicable même sur les ontologies qui ne disposent pas des instances au préalable.
- En outre, nous comptons tester d'autres techniques de classification pour la détection de mappings candidats à la fusion, tel que les réseaux de neurones.
- En plus, nous planifions d'étudier et d'analyser d'autres types de connaissances représentées dans les ontologies sources, pour augmenter la précision des résultats de l'algorithme de détection de mappings candidats à la fusion, tel que les sémantiques représentées par les axiomes dans les ontologies sources.

- Enfin, nous planifions d’étudier et d’appliquer d’autres stratégies d’optimisation pour améliorer les performances du système global.
- …

Bibliographie

Alani, H., & Brewster, C. (2005). Ontology ranking based on the analysis of concept structures. In Proceedings of the 3rd international conference on Knowledge capture pp. 51–58.

Amrouch S., Khadir M.T. (2009). Fusion d'ontologies de domaine pour la gestion de connaissances d'une turbine à vapeur, 2nd International Conference in Applied Informatics, (ICAI 2009), Bordj Bouariridj, Algérie.

Amrouch, S., Chefrour, A., Souici, L. (2011). Decision Trees for handwritten Arabic words recognition. International Arabe Conference on Information Technologies (ACIT'2011), NAUSS University, Riyadh, Arab Saudite.

Amrouch, S., Mostefai, S., (2012a). Ontology Interoperability techniques, the state of the art. International Journal of Information Organization, 2(1), pp. 20-27.

Amrouch, S., Mostefai, S., (2012b). Survey on the literature of ontology Mapping, Aligning and Merging. International Conference on Information Technologies and e-Services, (ICITeS'2012), Sousse, Tunisia.

Amrouch, S., Mostefai, S., (2012c). An Architecture for semi-automatic Ontology Merging system, In Proceedings of the International Arab Conference on Information Technologies, (ACIT'2012), Zarqua University, Amman, Jordan.

Amrouch, S., Mostefai, S., (2012d). A new Algorithm for Fully Automated Ontology Merging based on semantics using WordNet', International Journal of Information Studies, 4(1), pp. 37-47.

Amrouch, S., Mostefai, S., (2012e). Syntactico-Semantic Algorithm for Automatic Ontology Merging, In Proceedings of the International Conference of Information Technologies and e-Services (ICITeS'2012), Sousse, Tunisia.

Amrouch, S., Mostefai, S., (2013). Semantic Integration for Automatic Ontology Mapping, In Proceedings of the International Conference on Advanced Information Technologies and Applications (ICAITA'2013), Dubai, United Arab Emirates.

Amrouch, S., Mostefai, S. and Fahad, M., (2016). Decion Trees for Automatic Ontology Matching, International Journal of Meatadata, Semantics and Ontologies (IJMSO), 11(3), pp. 180-190.

Arpírez, J. C., Corcho, O., Fernández-López, M., & Gómez-Pérez, A. (2001). WebODE: a scalable workbench for ontological engineering. In Proceedings of the 1st international conference on Knowledge capture.Victoria, Canada .pp. 6–13.

Aubry, S. (2007). Annotations et gestion des connaissances en environnement virtuel collaboratif. Thèse de doctorat, Université de Technologie de Compiègne.

Baader, F., Calvanese, D., McGuinness, D., Nardi, D., & Patel-Schneider, P. (2003). The description logic handbook: Theory, implementation and applications. Cambridge university press, New York, NY, USA, ISBN:0-521-78176-0.

Bachimont, B. (2000). Engagement sémantique et engagement ontologique : conception et réalisation d'ontologies en Ingénierie des connaissances. In J. Charlet, M. Zacklad, G. Kassel & D. Bourigault (Eds.), Ingénierie des connaissances, évolutions récentes et nouveaux défis. Paris: Eyrolles, pp. 305–323.

Bachimont, B. (2001). Modélisation linguistique et modélisation logique des ontologies: l'apport de l'ontologie formelle. In Actes de la 12ème conférence Francophone Ingénierie des Connaissances, Plate-Forme AFIA, Grenoble, France, pp. 349–368.

Bechhofer, S., Horrocks, I., Goble, C., & Stevens, R. (2001). OilEd: a reasonable ontology editor for the semantic web. In Proceedings of the Joint 24th-German/9th-Austrian Conference KI 2001: Advances in Artificial Intelligence conference, Vienna, Austria, pp. 396–408.

Borst, W. N. (1997). Construction of engineering ontologies for knowledge sharing and reuse. Thèse de Doctorat, Université de Twente, Hollande.

Bouillon, P. (1998). Traitement automatique des langues naturelles. Livre, Aupelf - Uref, Edition Duclot, ISBN 2-8011-1181-3, De Boeck Université. Bruxelles.

Bouquet, P., & Serafini, L. (2001). Two formalizations of context: a comparison. In Modeling and Using Context, Proceedings of the Third International and Interdisciplinary Conference, CONTEXT Dundee, UK, pp. 87–101.

Brewster, C., Alani, H., Dasmahapatra, S., & Wilks, Y. (2004). Data driven ontology evaluation. In Proceedings of the 4th International Conference on Language Resources and Evaluation, Lisbon, Portugal, pp. 641-644.

Brodeur, J. (2004). Interopérabilité des données géospatiales: élaboration du concept de proximité géosémantique. Thèse de doctorat, Université Laval, Canada.

Bateman, J. A., Kasper, R. T., Moore, J. D., & Whitney, R. A. (1990). A general organization of knowledge for natural language processing: the penman upper model. Technical report, USC/Information Sciences Institute, Marina del Rey, CA.

Benjamins, R., Contreras, J., Corcho, O., & Gomez-Perez, A. (2002). The six challenges of the Semantic Web. In KR2002 Semantic web workshop.

Blázquez, J., Fernández, M., García-Pinar, J. M., & Gómez-Pérez, A. (1998). Building ontologies at the knowledge level using the ontology design environment. In Proceedings of the 11th Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW 98), Banff, Canada.

Borst, P., Akkermans, H., & Top, J. (1997). Engineering ontologies. International Journal of Human-Computer Studies, 46(2-3), pp. 365–406.

Bruijn J., Ehrig M., Feier C., Martín-Recuerda F., Scharffe F., et Weiten M. (2006). Ontology mediation, merging and aligning. », Semantic web technologies, pp.95-113.

Carroll, J. J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A. and Wilkinson, K. (2003). "Jena: Implementing the Semantic Web Recommendations", Technical report, Digital Media Systems Laboratory, HP Laboratories, Bristol.

Chaumartin, F.-R. (2007). WordNet et son écosystème: un ensemble de ressources linguistiques de large couverture. In Proceeding of the Colloque BD lexicales. Montréal, Canada. pp. 59-78.

Cho, M., Kim, H., & Kim, P. (2006). A new method for ontology merging based on concept using wordnet. In Proceedings of the 8th International Conference on Advanced Communication Technology, ICACT 2006. pp. 1573–1576.

Choi, N., Song, I. Y., & Han, H. (2006). A survey on ontology mapping. ACM Sigmod Record, 35(3), p. 34–41.

Cornuéjols, A., Miclet, L., & Kodratoff, Y. (2010). Apprentissage artificiel-Concepts et algorithmes. 2ème édition du Livre, éditions Eyrolles, ISBN-10: 2212124716.

D' Aquin, M., Gridinoc, L., Angeletou, S., Sabou, M., & Motta, E. (2007). Watson: A gateway for next generation semantic web applications. In Proceedings of the 4th European Semantic Web Conference, ESWC 2007. Innsbruck, Austria.

Deere, J. (2005). Ontology Definition Metamodel. In Book, Model Driven Architecture and Ontology Development, Part 2, Print ISBN 978-3-540-32180-4, Springer Berlin Heidelberg, pp. 181-199.

Dieng, R., Corby, O., Giboin, A., Golebiowska, J., Matta, N., & Ribiere, M. (2000). Méthodes et outils pour la gestion des connaissances, une approche pluridisciplinaire du Knowledge Management, INFORMATIQUES Série Systèmes d'information , 2ème édition, (Vol. 2). ISBN 2 10 006300 6. Dunod.

Djeddi, W.E. and Khadir, M.T. (2013). Ontology alignment using artificial neural network for large-scale ontologies, Int. J. Metadata, Semantics and Ontologies, Vol. 8, No. 1, pp.75–92.

Doan, A. H., Madhavan, J., Domingos, P., & Halevy, A. (2004). Ontology matching: A machine learning approach. In Handbook on Ontologies in Information Systems, pp. 397–416, Springer Berlin Heidelberg.

Domingue, J. (1998). Tadzebao and WebOnto: Discussing, browsing, and editing ontologies on the web. In Proceedings of the 11th Knowledge Acquisition for Knowledge-Based Systems Workshop. Banff, Canada.

Dou, D., McDermott, D., & Qi, P. (2005). Ontology translation by ontology merging and automated reasoning. Chapter 4 In book Ontologies for Agents: Theory and Experiences. pp. 73-94, Birkhäuser Basel , doi: 10.1007/3-7643-7361-X_4.

Dou D., Mcdermott D., et Qi P. (2002). Ontology translation by ontology merging and automated reasoning. In Proc. EKAW2002 Workshop on Ontologies for Multi-Agent Systems, 3 18. Yale University. http://dl.acm.org/citation.cfm?id=1087067.

Dowling, W.F., Gallier, J.H. (1984). R., Kenepa, B., & Benjamins, V. R. (2000). Wonder Tools? A comparative study of ontological engineering tools. International Journal of Human-Computer Studies, 52(6), pp.1111–1133.

Duyhoa, N., &Bellahsene, Z. (2013). YAM++ results for OAEI 2013', Proceedings Of the 8thWorkschop on Ontology Mapping (OM). Sydney, Australia, pp. 211-218.

Ehrig, M., Haase, P., Stojanovic, N., & Hefke, M. (2004). Similarity for ontologies-a comprehensive framework. In Workshop Enterprise Modelling and Ontology: Ingredients for Interoperability, PAKM (Vol. 2004).

Euzenat, J., Mocan, A., & Scharffe, F. (2008). Ontology alignments. Chapter 6. In book Ontology Management, Springer US, doi: 10.1007/978-0-387-69900-4_6, pp. 177–206.

Euzenat, J., & Valtchev, P. (2004). Similarity-based ontology alignment in OWL-lite. In Proceedings of the 16th European Conference on Artificial Intelligence, ECAI (Vol. 16, p. 333). Valencia, Spain, publié par: Ramon López de Mántaras, Lorenza Saitta, IOS press.

Fahad, M., & Qadir, M. A. (2008). A framework for ontology evaluation. In Proceedings of the 21st International Conference on Conceptual Structures (ICCS), Toulouse, France, pp. 7–11.

Fahad, M., Qadir, M. A., & Noshairwan, M. W. (2007). Semantic Inconsistency Errors in Ontology. In Proceeding of the 4th International Conference on Granular Computing, GRC 2007. IEEE, California, USA, pp. 283–293.

Fahad, M. (2015). DKP-AOM: results for OAEI 2015. In Proceedings Of the 10thWorkschop on Ontology Mapping (OM),Bethlehem, US.

Farquhar, A., Fikes, R., & Rice, J. (1997). The ontolingua server: A tool for collaborative ontology construction. International journal of human-computer studies, 46(6), pp. 707–727.

Fellbaum, C. (2010). WordNet. Chapter 10 In Book Theory and Applications of Ontology: Computer Applications, Springer Netherlands, doi: 10.1007/978-90-481-8847-5_10pp. 231–243.

Fernandez-Lopez, M., & Corcho, O. (2010). Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web. livre Springer Publishing Company, ISBN:1846283965 .

Fernández-López, M., Gómez-Pérez, A., & Juristo, N. (1997). METHONTOLOGY: From Ontological Art Towards Ontological Engineering. In proceeding of the 6th Spring Symposium on Ontological Engineering. AAAI-97, Stanford university, California, USA, AAAI Press. pp. 33-40.

Fu, H., & Nguifo, E. (2004). A parallel algorithm to generate formal concepts for large data. In book Concept Lattices, pp. 394-401. Springer Berlin Heidelberg.

Furst, F. (2004). Contribution à l'ingénierie des ontologies: une méthode et un outil d'opérationnalisation. Thèse de doctorat, Université de Nantes, France.

Fürst, F. (2002). L'ingénierie ontologique. Rapport de recherche, Institut de Recherche en Informatique de Nantes. France.

Gaëlle, L. (2002). État de l'art Ontologies et Intégration/Fusion d'ontologies (Rapport de stage). France: Laboratoire Dialogue et Intermédiations Intelligentes de la direction des interactions Humaines DIH/D2I, centre de recherche et de développement de France Télécom, Lannion. France.

Gandon, F. (2002). Distributed Artificial Intelligence and Knowledge Management: ontologies and multi-agent systems for a corporate semantic web. Thèse de doctorat, INRIA and University of Nice - Sophia Antipolis, France.

Gangemi, A., Guarino, N., Masolo, C., & Oltramari, A. (2003). Sweetening wordnet with DOLCE. AI magazine, 24(3), pp. 13-24.

Gangemi, A., Pisanelli, D., & Steve, G. (1998). Ontology integration: Experiences with medical terminologies. In Formal ontology in information systems. (Vol. 46). pp. 98–94) . IOS Press, Amsterdam, AM.

Genesereth M-R., Fikes R-E.. (1992). « Knowledge interchange format-version 3.0: reference manual. », Technical Report Logic. Computer Science Department, Stanford University, San Francisco, CA. http://www.cs.auckland.ac.nz/courses/compsci367s2c/resources/kif.pdf.

Ghidini C. et Giunchiglia F.. (2003). A semantics for abstraction. In Proceedings of the 16th European conference on Artificial Intelligence, pp.343-347. University of Trento. http://eprints.biblio.unitn.it/496/.

Gómez-Pérez, A. (1995). Some ideas and examples to evaluate ontologies. In Proceedings of the 11th Conference on Artificial Intelligence for Applications, Los Angeles, California, Canada. pp. 299–305.

Gómez-Pérez, A., Fernández, M., & De Vicente, A. (1996). Towards a method to conceptualize domain ontologies. In Proceeding of the 12th European Conference on Artificial Intelligence (ECAI'96), Budapest, Rumanía. pp. 41-52.

Gómez-Pérez A., Fernandez-Lopez M., et Corcho O. (2006). Ontological Engineering: With Examples from the Areas of Knowledge Management, E-Commerce and the Semantic Web. First Edition. Springer Science & Business Media.

Gómez-Pérez, A. (1999a). Ontological engineering: A state of the art. Expert Update: Knowledge Based Systems and Applied Artificial Intelligence, 2(3), Elsevier, pp. 33–43.

Gómez-Pérez, A. (1999b). Evaluation of taxonomic knowledge in ontologies and knowledge bases. In Proceedings of the 12th Banff Knowledge Acquisition for Knowledge-Based Systems, KAW'99, Banff, Alberta, Canada, pp. 6.1.1-6.1.18.

Gruber T. (1993). A translation approach to portable ontology specifications. Knowledge Acquisition 5 (2): pp.199-220. doi:10.1006/knac.1993.1008.

Gruninger, M., & Fox, M. S. (1995). Methodology for the Design and Evaluation of Ontologies. In *Proceedings of the Workshop on Basic Ontological Issues in Knowledge Sharing, in 14th International Joint Conference on Artificial Intelligence, IJCAI'95, Montreal, Canada.* pp. 6.1-6.10.

Gruninger, M. (1996). Designing and evaluating generic ontologies. In Proceedings of the 12th European Conference of Artificial Intelligence pp. 53–64.

Guarino, N. (1995). Formal ontology, conceptual analysis and knowledge representation. International Journal of Human Computer Studies, 43(5), Elsevier, pp. 625–640.

Guarino, N. (1997). Semantic matching: Formal ontological distinctions for information organization, extraction, and integration. In Information Extraction A Multidisciplinary Approach to an Emerging Information Technology. In Proceedings of the International Summer School, SCIE-97 Frascati, Italy, pp. 139–170.

Guzmán-Arenas A. et Cuevas A-D. (2010). Knowledge accumulation through automatic merging of ontologies. Expert Systems with Applications 37 (3). pp. 1991-2005.

Hakimpour, F., & Geppert, A. (2001). Resolving semantic heterogeneity in schema integration. In Proceedings of the international conference on Formal Ontology in Information Systems-Vol (2001). pp. 297–308.

Hamming, R. W. (1950). Error detecting and error correcting codes. Bell System technical journal, 29(2), pp.147–160.

Hendler, J., Lassila, O., & Berners-Lee, T. (2001). The semantic web. Scientific American, Press Room, 284(5), pp. 34–43.

Holmes, G., Donkin, A. and Witten, I. H. (1994) Weka: A machine learning workbench. In Proceedings of the 2nd Australian and New Zealand Conference on Intelligent Information Systems, Brisbane, Australia. pp. 357-361.

Horridge, M., Knublauch, H., Rector, A., Stevens, R., & Wroe, C. (2004). A Practical Guide To Building OWL Ontologies Using The Protégé-OWL Plugin and CO-ODE Tools. Edition 1.0. University of Manchester.

Hunter, J., Drennan, J., & Little, S. (2004). Realizing the hydrogen economy through semantic web technologies. Intelligent Systems Journal, IEEE, 19(1), pp. 40–47.

Jaro, M. A. (1989). Advances in record-linkage methodology as applied to matching the 1985 census of Tampa, Florida. Journal of the American Statistical Association, 84(406), pp. 414–420.

Jean, S. (2007). OntoQL, un langage d'exploitation des bases de données à base ontologique. Thèse de Doctorat, Université de Poitiers, France.

Jian, N., Hu, W., Cheng, G., & Qu, Y. (2005). Falcon-ao: Aligning ontologies with falcon. In Proceedings of the Knowledge CAPture Workshop on Integrating Ontologies, K-CAP, 2005, Banff, Alberta, Canada, pp. 85–91.

Jiménez-Ruiz, E. and Grau, B.C. (2011) Logmap: logic-based and scalable ontology mapping. In Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N.F. and Blomqvist, E. (Eds): International Semantic Web Conference, Springer, pp. 273–288.

Jiménez-Ruiz, E.,Grau, B.C.,Solimando, A., Cross, V. (2015). LogMap family results for OAEI 2015. In Proceedings Of the 10thWorkschop on Ontology Mapping (OM),Bethlehem, US,pp. 126-134.

Jung, J., & Euzenat, J. (2007). Towards semantic social networks. In The Semantic Web: Research and Applications, In Proceedings of the 4th European Semantic Web Conference, ESWC 2007, Innsbruck, Austria, pp. 267–280.

Jung, J., Zimmerman, A., & Euzenat, J. (2007). Concept-based query transformation based on semantic centrality in semantic peer-to-peer environment. In book Advances in Data and Web Management, Springer Berlin Heidelberg.doi: 10.1007/978-3-540-72524-4_64. pp. 622–629.

Karen Sparek, J. (2004). A statistical interpretation of term specificity and its specification in retreival. journal of documentation. 60(5), p. 493-502.

Klein, M. (2001). Combining and relating ontologies: an analysis of problems and solutions. In Proceedings of the Workshop on ontologies and information sharing, IJCAI'01,Seattle, USA. Vol. 1, pp. 53–62.

Klein, M., & Fensel, D. (2001). Ontology versioning on the Semantic Web. In Proceedings of the International Semantic Web Working Symposium (SWWS), Stanford University, California, USA. pp. 75–91.

Knight, K., Chander, I., Haines, M., Hatzivassiloglou, V., Hovy, E., Iida, M., Yamada, K. (1995). Filling knowledge gaps in a broad-coverage machine translation system. In Proceedings of the 14th joint conference on Artificial Intelligence (IJCAI'95), Montreal, Quebec, Canada. pp. 1390-1396.

Knight, K., & Luk, S. K. (1994). Building a large-scale knowledge base for machine translation. In Proceedings of the 12th National Conference on Artificial Intelligence. Seattle, Washington, USA. p. 773–773.

Kobayashi, M., & Takeda, K. (2000). Information retrieval on the web. ACM Computing Surveys (CSUR), 32(2), ACM New York, NY, USA, pp. 144–173.

Kong, H., Hwang, M., & Kim, P. (2005). A new methodology for merging the heterogeneous domain ontologies based on the wordnet. In the proceedings of

the International Conference on Next Generation Web Services Practices, NWeSP' 2005, Seoul, Korea. pp. 235-240.

Kotis, K., Vouros, G. A., & Stergiou, K. (2006). Towards automatic merging of domain ontologies: The HCONE-merge approach. In journal of Web Semantics: Science, Services and Agents on the World Wide Web, 4(1), Elsevier. pp.60–79.

Levenshtein, V. I. (1966). Binary Coors Capable Or Correcting Deletions, Insertions, And Reversals. In journal of Soviet Physics-Doklady , 10(8). pp. 707-710.

Li, S., Fan, H., Wang, Y., & Hong, L. (2009). A Model and Approach for Heterogeneous Ontology Automatic Merging. In Proceedings of the 4th International Joint Conference on Computational Sciences and Optimization, CSO'09.Sanya, Hainan Island, Chaina. (Vol. 2). pp. 214–217.

López, M. F., Gómez-Pérez, A., Sierra, J. P., & Sierra, A. P. (1999). Building a chemical ontology using methontology and the ontology design environment. In journal of Intelligent Systems and their Applications, IEEE, 14(1), pp. 37–46.

Madhavan, J., Bernstein, P. A., & Rahm, E. (2001). Generic schema matching with cupid. In Proceedings of the 27th International Conference on Very Large Data Bases, Roma, Italy, pp. 49–58.

Maedche, A., & Staab, S. (2001). Ontology learning for the semantic web. In journal of Intelligent Systems, IEEE, 16(2), pp. 72–79.

Maedche, A., Staab, S. (2002) 'Measuring similarity between ontologies', Knowledge engineering and knowledge management: Ontologies and the semantic web, Springer Berlin Heidelberg, pp. 251-263.

Maedche A., Motik B., Stojanovic L., Studer R., et Volz R. (2003). « An Infrastructure for Searching, Reusing and Evolving Distributed Ontologies. » In Proceedings of the 12th International Conference on World Wide Web, pp.439-448. WWW '03. New York, NY, USA: ACM. doi:10.1145/775152.775215.

Magnini, B., & Speranza, M. (2002). Merging global and specialized linguistic ontologies. In Proceedings of the Workshop Ontolex-2002 Ontologies and Lexical Knowledge Bases, LREC-2002, Las Palmas, pp. 43-48.

Maiz, N., Boussaid, O., & Bentayeb, F. (2008). Fusion automatique des ontologies par classification hiérarchique pour la conception d'un entrepôt de données. 2ème Journées Francophones sur les Ontologies, (JFO'08), Lyon, France. pp. 17-27.

Maiz, N., Fahad, M., Boussaid, O., & Bentayeb, F. (2010). Automatic Ontology Merging by Hierarchical Clustering and Inference Mechanisms. In Proceeding of the 10th International Conference on Knowledge Management and Knowledge Technologies (I-KNOW 2010), Graz, Austria. pp.81-93.

Melnik, S., Garcia-Molina, H., & Rahm, E. (2002). Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In Proceedings of the 18th International Conference on Data Engineering, San Jos, California, USA. pp. 117–128.

Miller, G. A., Beckwith, R., Fellbaum, C., Gross, D., & Miller, K. J. (1990). Introduction to wordnet: An on-line lexical database. International journal of lexicography, 3(4), Oxford univ press, pp. 235–244.

Mitchell, T (1997). Machine Learning, Chapter 3 du livre, pp.67. WBC/McGraw-Hill

Mitra, P., & Wiederhold, G. (2001). An algebra for semantic interoperability of information sources. In Proceedings of the 2nd International Symposium on Bioinformatics and Bioengineering, Bethes da Maryland. IEEE, pp. 174–182.

Mitra, P., Wiederhold, G., & Kersten, M. (2000). A graph-oriented model for articulation of ontology interdependencies. Advances in Database Technology, Proceedings of the 7th International Conference on Extending DataBase Technology, EDBT'00, Konstanz, Germany, pp. 86–100.

Mitra P. et Wiederhold G. (2002). Resolving terminological heterogeneity in ontologies. » In Proceedings of the workshop on Ontologies and Semantic Interoperability, 15th ECAI, Lyon, France, pp.45-50.

Mizoguchi, R. (2003). Part 1: introduction to ontological engineering. New Generation Computing, 21(4), Springer. pp. 365–384.

Mohsenzadeh, M., Shams, F., & Teshnehlab, M. (2005). A new approach for merging ontologies. Enformatica Transactions on Engineering, Computing and Technology, pp. 153–159.

Mostefai, S. (2008). Fusion d'ontologies dans une optique PLM. In Proceedings du premier Congrès des Innovations Mécaniques, CIM'08, Sousse, Tunisie.

Neeches, R., Fikes, R. E., Finin, T., Gruber, T., Patil, R., Senator, T., & Swartout, W. R. (1991). Enabling technology for knowledge sharing. AI magazine, 12(3), AAAI Press. pp. 36-56.

Ngo, D., (2012). Enhancing ontology matching by using machine learning, graph matching and information retrieval techniques, In PhD. Thesis, University Montpellier II.

Niles, I., & Pease, A. (2001). Towards a standard upper ontology. In Proceedings of the 2nd international conference on Formal Ontology in Information Systems, Ogunquit, ME, USA ,pp. 2–9.

Noshairwan, W., Qadir, M., & Fahad, M. (2007). Sufficient Knowledge Omission error and Redundant Disjoint Relation in Ontology. In Advances in Intelligent Web Mastering, Proceedings of the 5th Atlantic Web Intelligence Conference – WIC'2007, Fontainebleau, France. pp. 260–265.

Noy, N. F., & Hafner, C. D. (1997). The state of the art in ontology design: A survey and comparative review. AI magazine, 18(3), AAAI Press. pp. 53-74.

Noy, N. F. (2004). Semantic integration: a survey of ontology-based approaches. SIGMOD record, 33(4), pp. 65–70.

Noy, N. F., & Musen, M. A. (1999). An algorithm for merging and aligning ontologies: Automation and tool support. In Proceedings of the Workshop on Ontology Management at the 16th National Conference on Artificial Intelligence (AAAI-99), Orlando, Florida, pp. 17-27.

Noy, N. F., & Musen, M. A. (2000). Algorithm and tool for automated ontology merging and alignment. In Proceedings of the 17th National Conference on Artificial Intelligence (AAAI-00). Austin, Texas, USA. pp. 450-455.

Noy, N. F., & Musen, M. A. (2001). Anchor-PROMPT: Using non-local context for semantic matching. In Proceedings of the workshop on ontologies and information sharing at the international joint conference on artificial intelligence , IJCAI'01,Seattle, Wachington, USA, pp. 63–70.

Noy, N. F., & Musen, M. A. (2002). Evaluating ontology-mapping tools: Requirements and experience. In Proceedings of the Workshop on Evaluation of Ontology Tools at 13th International Conference On Knowledge Engineering and Knowledge Management, EKAW'02, Siguenza, Spain, (Vol. 2), pp. 1-14.

Noy, N. F., & Musen, M. A. (2003). The PROMPT suite: interactive tools for ontology merging and mapping. International Journal of Human-Computer Studies, Elsevier, 59(6), pp. 983–1024.

Noy, N. F., & Musen, M. A. (2003). Ontology Versioning as an Element of an Ontology-Management Framework. Journal of Intelligent Systems, IEEE, 19(4), pp.6-13.

Osório, F. S. (1998). INSS: Un système hybride neuro-symbolique pour l'apprentissage automatique constructif. Thèse de doctorat, Institut National Polytechnique de Grenoble-INPG. France.

Pease, A., Niles, I., & Li, J. (2002). The suggested upper merged ontology: A large ontology for the semantic web and its applications. In Working Notes of the AAAI-2002 Workshop on Ontologies and the Semantic Web (Vol. 28). Edmonton, Canada.

Pinto, H. S. (1999). Towards ontology reuse. In Proceedings of the AAAI99's Workshop on Ontology Management at the 16th National Conference on Artificial Intelligence, WS-99 , Orlando, Florida, pp. 67-73.

Pinto, H. S., Gómez-Pérez, A., & Martins, J. P. (1999). Some issues on ontology integration. In Proceedings of the WorkShop on Ontolgies and Problem Solving Methods: Lessons learned and Future trends at the 16th International Joint Conference on Artificial Intelligence, IJCAI'99, Stockholm, Sweden, pp. 7.1-7.12.

Pinto, H. S., & Martins, J. P. (2000). Reusing ontologies. In Proceedings of the Spring Symposium on Bringing Knowledge to Business Processes, Karlsruhe, Germany: AAAI. (Vol. 2), pp. 77-84.

Porzel, R., & Malaka, R. (2004). A task-based approach for ontology evaluation. In Proceedings of the 16th European Conference on Artificial Intelligence, ECAI'04, Workshop on Ontology Learning and Population, Valencia, Spain. pp. 316-321.

Predoiu, L., Feier, C., Scharffe, F., De Bruijn, J., Martın-Recuerda, F., Manov, D., & Ehrig, M. (2005). State-of-the-art survey on Ontology Merging and Aligning V2 (SEKT project, EU-IST Integrated Project IST, Delivrable N° D4.2.2), DERI Innsbruck, pp. 1-121.

Qadir, M. A., Fahad, M., & Shah, S. A. H. (2007). Incompleteness Errors in Ontology. In Proceedings of the 3rd International Conference on Granular Computing, GRC 2007. IEEE, Silicon, Valley, California, USA. pp. 279–282.

Ragot, N. (2003). MELIDIS: Reconnaissance de formes par modélisation mixte intrinsèque/discriminante à base de système d'inférence floue hiérarchisés. Thèse de doctorat, Université Rennes1. France.

Rahm, E., & Bernstein, P. A. (2001). A survey of approaches to automatic schema matching. the VLDB Journal, Springer, 10(4), pp. 334–350.

Raunich S., et Rahm E. (2011). ATOM: Automatic target-driven ontology merging. In Proceedings of the 27th International Conference on Data Engineering,(IEEE, ICDE, 2011).

Raunich S., et Rahm E. (2012). Towards a benchmark for ontology merging. In Proceedings of the 7th International Conference On the Move to Meaningful Internet Systems, OTM Workshop on Enterprise Integration, Interoperability and Networking (EI2N'12 ,Springer), Rome, Italy, pp.124-133..

Richardson B. et Mazlack L-J. (2004). Approximate ontology merging for the semantic Web. In Proceedings of the Annual Meeting of the Fuzzy Information processing NAFIPS '04. IEEE, vol (2). pp.641-646.

Robin, C., & Uma, G. (2010). A novel algorithm for fully automated ontology merging using hybrid strategy. European Journal of Scientific Research, Scientific Research Platform (SRP) , 47(1), pp. 74–81.

Saïd, I. (2006). INTEGRATION DES SYSTEMES D'INFORMATION INDUSTRIELS, Une approche flexible basée sur les services sémantiques. Thèse de Doctorat de l'Ecole Nationale Supérieure des Mines de Saint-Etienne. France.

Saleem, A. (2006). Semantic Web Vision: survey of ontology mapping systems and evaluation of progress. In Clerk Maxwell Journal, ed. A Treatise on Electricity and Magnetism, Blekinge Institute of Technology, Sweden,3rd edition, vol(2), Clarendon, Oxford, pp. 68–73.

Scharffe, F. (2007). Dynamerge: A merging algorithm for structured data integration on the web. In Proceeedings of the DASFAA International Workshop on Scalable Web Information Integration and Service (SWIIS 2007).Bangkok, Thailand, pp. 85-94.

Scharffe, F., & De Bruijn, J. (2005). A language to specify mappings between ontologies. In Proceedings of the 1st International Conference on Signal-Image Technology and Internet-Based Systems (SITIS2005), IEEE Conference, Yandoué, Cameroon, pp. 267-271.

Shvaiko, P., & Euzenat, J. (2005). A survey of schema-based matching approaches. Journal of Data Semantics IV, Springer Verlag Berlin Heidelberg, doi: 10.1007/11603412_5 pp. 146–171.

Sergey, M., Hector, G-M., Erhard, R. (2002). Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In ICDE, pp. 117–128.

Skuce, D.& READER, N. T. (1997). How We Might Reach Agreement on Shared Ontologies: A Fundamental Approach, In AAAI Spring Symposium Series, Workshop on Ontological Engineering. Stanford University, California, USA, pp. 114-119.

Snelting, G. (1994). Reengineering of configurations based on mathematical concept analysis. In Proceedings of the 16th International Conference on Software Engineering. Sorrento, Italy, PP. 146-189.

Sowa, J. F. (1995). Top-level ontological categories. International journal of human-computer studies, 43(5), Elsevier. pp. 669–685.

Sowa, J. F. (1997). Electronic communication in the onto-std mailing list.

Stuckenschmidt, H., & Klein, M. (2004). Structure-based partitioning of large concept hierarchies. The Semantic Web, Proceedings of the 3rd International Semantic Web Conference, Hiroshima, Japan–ISWC 2004, pp. 289–303.

Studer R., Benjamins V., Fensel D. (1998). Knowledge Engineering: Principles and Methods". In Data and Knowledge Engineering.

Stumme, G., & Maedche, A. (2001). FCA-Merge: Bottom-up merging of ontologies. In Proceedings of the 17th International joint conference on artificial intelligence (Vol. 17), Seattle, Wachington, USA, pp. 225–234.

Sure, Y., Erdmann, M., Angele, J., Staab, S., Studer, R., & Wenke, D. (2002). OntoEdit: Collaborative ontology development for the semantic web. In Proceedings of the 1st International Semantic Web Conference , ISWC 2002, Sardinia, Italia. pp. 221–235.

Swartout B., Patil R., Knight K., Russ T. (1997). Use of Large Scale Ontologies. Spring Symposium Series on Ontological Engineering. Stanford University, CA, pp. 138-148.

Tang, J., Li, J., Liang, B., Huang, X., Li, Y., & Wang, K. (2006). Using Bayesian decision for ontology mapping. In journal of Web Semantics: Science, Services and Agents on the World Wide Web, 4(4), Elsevier, pp. 243–262.

Uschold, M., & King, M. (1995). Towards a methodology for building ontologies. In Proceedings of the 14th International Joint Conference on Artificial Intelligence, IJCAI'95, *Workshop on basic ontological issues in knowledge sharing. Montreal, Canada, pp. 15-30.*

Uschold, M., & Gruninger, M. (1996). Ontologies: Principles, methods and applications. Knowledge engineering review, Cambridge University Press New York, NY, USA. 11(2), pp. 93–136.

Uschold, M., & Gruninger, M. (2004). Ontologies and semantics for seamless connectivity. ACM SIGMod Record, 33(4), pp. 58–64.

Van Hage, W., Katrenko, S., & Schreiber, G. (2005). A method to combine linguistic ontology-mapping techniques. The Semantic Web –Proceedings of the 4th International Semantic Web Conference, ISWC 2005, Galway, Ireland, pp. 732–744.

Vizcaíno, A., Anquetil, N., Oliveira, K., Ruiz, F., & Piattini, M. (2005). Merging Software Maintenance Ontologies: Our Experience. In Proceedings of the 1st Workshop on Ontology, Conceptualizations and Epistemology for Soft-2. Using Ontologies in Software Engineering and Technology , ONTOSE'05, Alcala de Henares, Spain.

Voorhees, E. M. (1994). Software agents for information retrieval. In Working notes of the AAAI spring symposium on Software agents . Palo, Alto, California, USA. pp. 126-129.

Waralak, D. et Siricharoen, V. (2007). Ontologies and Object models in Object Oriented Software Engineering. IAENG International Journal of Computer Science 33(1), pp.19 24.

Wiederhold, G. (1994a). Interoperation, mediation and ontologies. In Proceedings of the 5th International Conference on Computer Generation Systems, FGCS'94, Workshop on Heterogeneous Cooperative Knowledge-Bases. ICOT, Tokyo, Japan, pp. 33-48.

Wiederhold, G. (1994b). An algebra for ontology composition. In Proceedings of 1994 Monterey Workshop on Formal Methods , Monterey, California, USA, pp. 56-61.

Winkler, W. E. (2006). Overview of record linkage and current research directions. In Book Overview of Record Linkage and Current Research Directions. Volume 2006-2. City: Statistical Research Division, U.S. Census Bureau; 2006:44, pp. 1-44. Available at: http://www.census.gov/srd/papers/pdf/rrs2006-02.pdf. accessed in 2015-10-06.