REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR

ET DE LA RECHERCHE SCIENTIFIQUE

CENTRE UNIVERSITAIRE DE KHENCHELA

DIRECTION DE LA POST GRADUATION ET DE LA RECHERCHE SCIENTIFIQUE

Ecole Doctorale de Sciences et Technologies de l'Information

et de la Communication

N° d'ordre :

Série :

MEMOIRE

Présenté en vue de l'obtention du diplôme de :

Magistère en informatique

Option : Système d'information et de connaissances

Thème :

Proposition d'un Algorithme pour l'Alignement des Ontologies dans des Systèmes Distribués

Présenté par :

Lehis Saida

Soutenu publiquement le 28/06/2011

Devant le jury :

Président : Dr LAOUAR Med Ridda, *Maître de conférence - Université de Tebessa.*

Rapporteur : Dr SERIDI Hassina**,** *Maître de conférence - Université de Annaba.*

Examinateur : Dr MOKHATI Farid, *Maître de conférence - Université de Oum el Bouaghi*

Examinateur : Dr AMIRAT Abdelkrim, *Maître de conférence - Université de Souk Ahras*

Merci mon Dieu de m'avoir donné
la force, la patience et la volonté d'arriver au terme de ce travail.

Je dédie ce modeste travail

À la mémoire de mon père Ahmed

À Ma mère Fatma qui m'a tout donné

À mon mari wahid je lui dédie ce travail
qui n'aurait pu aboutir sans son soutien

Aux petits anges:
Amira malak , Adem et Asma.

Remerciements

Je souhaite remercier très vivement mon encadreur Dr SERIDI Hassina, non parce que c'est l'usage, mais pour m'avoir guidé dans mes travaux, conseillé avec professionnalisme et une très grande expertise, sans jamais compter son temps. Ses apports majeurs me permettent aujourd'hui de vous présenter cette thèse.

Je suis très reconnaissante envers le Docteur LAOUAR Med Ridda de me faire l'honneur de présider le jury de cette soutenance. J'exprime toute ma gratitude au Docteur MOKHATI Farid et au Docteur AMIRAT Abdelkrim pour accepter d'être mes examinateurs de thèse.

Je remercie mes parents de m'avoir inculqué le goût du savoir et de l'ambition.

Je remercie mon mari pour ses encouragements permanents, son aide précieuse et soutien sans faille.

Merci à mes chèrs frères, Azedine, Adel et Abdelbaki, et mes sœurs. Pour leur soutien moral.

Mes amitiés s'adressent à mes collègues de la promotion 2008 de l'Ecole Doctorale, Karima, Rime,…..

Merci à tous mes amis qui m'ont permis de me changer les idées quand j'en avais besoin, Amel, Madina, Hayet, Radja,…

Table des matières

Chapitre 3 : Architecture pair à pair

Contribution :

Chapitre 4: Proposition d'un algorithme d'alignement dans un environnement pair à pair

Chapitre 5 : Implémentation et Evaluation

Liste des figures :

Liste des Tableaux :

Introduction Générale :

Dans le cadre d'applications distribuées, l'évolution des récents développements des systèmes à base de connaissances fait apparaitre la nécessité de spécifier une interprétation commune des informations échangées. Cela peut être effectué en utilisant des standards ou en adaptons ces systèmes pour qu'ils interprètent les informations sans ambiguïté, on parle alors d'interopérabilité.

Ainsi, de nouveaux besoins ont émergé pour représenter les connaissances et manipuler leur sémantique. En premier lieu, les ontologies sont devenues très populaires, en occupant une place de choix dans le domaine de l'ingénierie des connaissances et ont prouvé leur efficacité pour la représentation des connaissances. Une ontologie est définie comme une spécification formelle et explicite d'une conceptualisation partagée [Gruber, 1993], avec les significations suivantes :

- Le mot « Formelle » se rapporte au fait que l'ontologie devrait être compréhensible par une machine ;
- Le mot « Explicite » signifie que le type des concepts utilisés, et les contraintes sur leur utilisation sont explicitement définis ;
- Le mot « Conceptualisation » signifie qu'un modèle abstrait des phénomènes est identifié par des concepts appropriés à ces phénomènes ;
- Le mot « Partagée » reflète que l'ontologie devrait capturer la connaissance consensuelle admise par les communautés.

La prise de conscience des potentialités immenses des applications des ontologies, a engendré la construction d'un nombre de plus en plus important d'ontologies par différents individus et par différentes organisations.

Du coup, ces ontologies qui ont été créées initialement, pour pallier au problème de l'hétérogénéité des données, sur le web par exemple, sont devenues elles mêmes sources d'hétérogénéité.

Les ontologies sont souvent différentes car elles ont été conçues dans la perspective d'atteindre des objectifs divers et décrivent parfois des domaines variés. Néanmoins, il n'est pas rare de constater l'existence d'informations communes entre ces ontologies.

Ce problème d'hétérogénéité des ontologies peut être surmonté grâce à l'alignement des ontologies qui consiste à identifier des relations entre éléments de différentes ontologies. Euzenat et ses collègues [Euzenat et al., 2004] identifient différents types de méthodes

d'alignement d'ontologies : terminologiques, structurelles, extensionnelles et sémantiques, qui proviennent de disciplines variées, telles que la fouille de données, les sciences du langage, les statistiques ou la représentation des connaissances.

Par ailleurs, Dans le domaine des réseaux, le paradigme Pair à Pair (P2P) est devenu un des axes les plus importants dans le domaine de l'informatique distribuée. Le pair à pair est défini comme étant une classe d'applications qui prennent avantage des ressources de stockage, de traitement processeur, du contenu et de la présence humaine de façon totalement distribuée. Les opérations d'accès à ces ressources décentralisées se font dans un environnement où les connexions sont instables et non contrôlées. Cela implique qu'un système pair à pair doit être autonome et indépendant de tout serveur central [Shirky, 2000].

Le pair à pair suscite beaucoup d'intérêt, et son utilisation connaît une évolution très rapide. Cela est dû principalement à la nature du pair à pair qui comporte plusieurs avantages, comme l'auto organisation, la tolérance aux pannes, l'autonomie, le dynamisme. En plus de ces caractéristiques, le pair à pair constitue actuellement, une infrastructure largement utilisée pour le partage des ressources et des données dans les réseaux informatiques.

Le domaine de l'alignement des ontologies est un des domaines les plus investis par la communauté scientifique et industrielle. Dans ce contexte des nombreuses solutions basées sur des correspondances sémantiques entre les pairs ont été développées. Différents procédés sont utilisés pour générer ces correspondances de manière plus ou moins automatique.

Face aux problèmes précédemment évoqués, un algorithme est présenter dans cette thèse pour faire des alignements entre les ontologies dans un réseau pair à pair, Étant donné un pair source et un pair cible, cet algorithme calcule l'alignement des concept parmi les concepts contenus dans les ontologies (source et cible). Il adopte une stratégie d'alignement basée sur l'exploitation des mesures de similarité floues (simples et combinée).

Ainsi, cette thèse s'organise de la manière suivante :

Le Chapitre 1: présente les différents points de vu sur l'ontologie, ensuite les différents éléments qui la composent et les besoins auxquels elle répond, et les différentes classifications et les principaux formalismes utilisés pour représenter une ontologie.

Le Chapitre 2: traite les applications de l'alignement des ontologies, ensuite, les techniques et les méthodes utilisées dans la littérature qui attaquent le problème de recherche de la similarité, de la dissimilarité ou de la correspondance entre deux entités en général,

qu'elles apparaissent dans des schémas, ou dans des ontologies représentées soit en RDF(S), soit en OWL, ainsi les approches qui emploient ces techniques seront présentées.

Le Chapitre 3: Présente quelques définitions précises de ce qu'est le pair à pair, ensuite, il présente les différentes caractéristiques, les objectifs et les différents types du pair à pair, ainsi que quelques travaux d'alignement des ontologies dans un système pair à pair.

Le Chapitre 4: présente l'algorithme proposé pour l'alignement des ontologies dans un environnement pair à pair, qui adopte une stratégie d'alignement basée sur l'exploitation des mesures de similarité floues simples et combinées.
Notre solution se divise en deux parties :
a) la coordination entre les pairs dans un environnement pair à pair.
b) et l'algorithme d'alignement (flou matcher) qui s'exécute au niveau de chaque pair dans cet environnement.

Le Chapitre 5: Finalement, une implémentation et une évaluation de l'algorithme d'alignement d'ontologies proposé sont présentées dans le chapitre cinq. L'implémentation de l'algorithme a été effectuée en Java et les évaluations ont été faites sur des bases de tests standards publiés à http://oaei.ontologymatching.org/.

La conclusion de ce mémoire synthétise cette thèse, nous présentons un bilan de notre travail. Nous discutons également des perspectives de ce travail.

Problématique

Dans le cadre des travaux de recherche sur l'intégration sémantique d'applications via l'utilisation d'ontologies, les utilisateurs peuvent disposer d'ontologies propres où ils ont définies les différents concepts à manipuler. Pour uniformiser, il est nécessaire d'identifier des mises en correspondance (ou «mappings») entre les concepts des différentes ontologies utilisées. Des travaux ont été menés pour découvrir de façon automatique des correspondances entre les ontologies, en se basant sur des différents types de méthodes d'alignement d'ontologies: terminologiques, structurelles, extensionnelles et sémantiques, qui proviennent de disciplines variées.

Cependant, le partage des données et des connaissances est souvent confronté à un problème majeur, à savoir, l'hétérogénéité existante entre les différentes sources. La diversité rend la recherche et le partage d'information une tâche complexe et difficile, spécialement dans un environnement en pleine expansion tel que le Web et un système hautement dynamique et autonome tel que le pair à pair.

L'objectif de notre travail est de permettre l'exploitation d'un système pair à pair, et l'étude des techniques de coopération entre les pairs, et de mettre en place une stratégie de partage de mappings. Une orientation qui nous proposons est l'application de la logique floue à la recherche des alignements entre ontologies.

Une évaluation des résultats de mapping peut être donnée en utilisant les tests Benchmark développé par l'équipe de la campagne de tests OAEI (Ontology Alignment Evaluation Initiative).

Chapitre 1 :
Les Ontologies

« Il y a une science qui étudie l'être en tant qu'être et les attributs qui les appartiennent essentiellement »

Aristote

Introduction:

Le terme « ontologie », construit à partir des racines grecques *ontos* (ce que existe, l'existant) et *logos* (le discours, l'étude), est un mot que l'informatique a emprunté à la philosophie au début des années 1990. En philosophie, l'Ontologie est une branche fondamentale de la Métaphysique, qui s'intéresse à la notion d'existence, aux catégories fondamentales de l'existant et étudie les propriétés les plus générales de l'être.

L'utilisation d'ontologies en informatique vise à intégrer une couche de connaissances aux systèmes afin de permettre des traitements élaborés de l'information qu'ils manipulent.

La conception d'ontologies est une tâche difficile qui nécessite la mise en place de procédés élaborés afin d'extraire la connaissance d'un domaine, manipulable par les systèmes informatiques et interprétable par les êtres humains

Dans ce chapitre, on va étudier les différents points de vu sur l'ontologie, ensuite les différents éléments qui la composent et les besoins auxquels elle répond, puis, nous citons les différentes classifications et les principaux formalismes utilisés pour représenter une ontologie.

I. Définitions :

Plusieurs définitions ont été adoptées pour décrire une ontologie parmi les quelles on peut citer :

Définition 1 : Neshes et ses collègues furent les premiers à en proposer une définition à savoir : « *une ontologie définisse les termes et les relations de base du vocabulaire d'un domaine ainsi que les règles qui indiquent comment combiner les termes et les relations de façon à pouvoir étendre le vocabulaire.* » [Neshes et al., 1991]

Définition 2 : En 1993, Gruber formule la définition suivante :

« *une ontologie est une spécification formelle et explicite d'une conceptualisation partagée.* » [Gruber, 1993]

Définition 3 : En 1997, Guarino, affine la définition de Gruber en énonçant que :

«*Les ontologies sont des spécification partielles et formelles d'une conceptualisation commune.*» [Guarino, 1997]

II. Composants d'une ontologie :

Une ontologie ne peut être construite que dans le cadre d'un domaine précis de la connaissance, parce que beaucoup de termes n'ont pas le même sens d'un domaine à un autre. Les connaissances traduites par une ontologie sont à véhiculer à l'aide des principaux éléments suivants:

II.1 Concept

II.2 Relation

II.3 Les rôles

II.4 Les fonctions

II.5 Les axiomes

II.6 Les instances

***II.1 Concept* :** qui peut représenter un objet, une idée. Un concept peut être divisé en trois parties:

II.1.1 Un terme: est un élément lexical qui permet d'exprimer le concept en langue naturelle. Il peut admettre des synonymes.

II.1.2 La notion: également appelée intension du terme, contient la sémantique du concept, exprimé en terme de propriétés et attributs, et de contraintes.

II.1.3 L'ensemble d'objets: appelé extension du concept, regroupe les objets manipulés à travers le concept; ces objets sont appelés instances du concept.

Propriétés sur les concepts :

a. La généricité : un concept est générique s'il n'admet pas une extension, exemple, la vérité.

b. L'identité : permet de conclure si deux entités sont identiques

c. La rigidité : un concept est rigide si toute instance de concept en reste instance dans tous les mondes possibles.

d. L'anti-rigidité : toute instance du concept est définie par son appartenance à l'extension d'un autre concept.

e. Propriétés portant sur deux concepts :

L'équivalence : deux concepts sont équivalents s'ils ont la même extension.

La disjonction : deux concepts sont disjoints (incompatibles) si leurs extension sont disjointes.

II.2 Relation :

Une relation permet de lier les instances de concepts, ou des concepts génériques. Elles sont caractérisées par un (ou plusieurs) terme(s) et une signature qui précise le nombre d'instances de concepts que la relation lie, leurs types et les ordres des concepts.

Propriétés sur les relations

1. Propriétés algébrique :
    a. La symétrie
    b. La réflexivité
    c. La transitivité.
2. La cardinalité : nombre de relations possibles entre les mêmes concepts.
3. Propriétés liant deux relations :
    a. L'incompatibilité : Deux relations sont incompatibles si elles ne peuvent lier les mêmes concepts.
    b. L'inverse : Deux relations binaires sont inverses l'une de l'autre, si lie deux instances I_1 et I_2, l'autre lie I_2 et I_1.
    c. L'exclusivité : deux relations sont exclusives, si quand l'une lie des instances de concepts, l'autre ne le fait pas.

II.3 Les rôles :

Selon Sowa, « un rôle caractérise une entité par quelque rôle qu'elle joue dans sa relation à une autre entité. Le type « Humain », par exemple, est un type de phénomène qui dépend de la forme interne de l'entité; mais la même entité peut être caractérisée par des rôles du type, Mère, Employé ou Piéton.» [Sowa, 2000]

II.4 Les fonctions :

Les fonctions sont aussi des cas particuliers de relations dans lesquelles le nième élément de la relation est défini à partir des n-premiers. Comme exemple de fonctions binaires il y a la fonction mère-de ou carré-de, comme fonction ternaire, le prix d'une voiture usagée sur lequel on peut se baser pour calculer le prix d'une voiture d'occasion en fonction de son modèle, de sa date de construction et de son kilométrage.

II.5 Les axiomes :

Les axiomes sont utiles à la structuration de phrases qui sont toujours vraies. Ils permettent de contraindre les valeurs de classes ou d'instances.

II.6 Les instances :

Les instances sont utilisées pour représenter des éléments dans un domaine.

III. Utilité des ontologies :

a) La communication : les ontologies peuvent intervenir dans la communication entre les humains. Elles permettent de créer un vocabulaire standard entre les membres d'une association, d'une entreprise. etc.

b) L'aide à la spécification de systèmes : l'ontologie permet de rendre les documents d'un processus plus compréhensibles.

c) L'interopérabilité : l'ontologie répertorie les concepts que les applications peuvent s'échanger, même si elles sont distantes et développées sur des bases différentes (Windows, linux,..)

d) L'indexation et la recherche d'information : les ontologies ont été utilisées dont les web sémantique pour fournir des index conceptuels décrivant les ressources sur le web.

IV. Typologie d'ontologie :

Selon Studer, "*il y a différents types d'ontologie et chaque type remplit un rôle différent dans le processus de construction du modèle du domaine*".[Studer, 1998]

Les ontologies peuvent être classées selon plusieurs dimensions, par exemple :

IV.1 Typologie selon l'objet de conceptualisation [Gomez-Perez, 1999], [Guarino, 1997], [Mizoguchi et al., 1998] ;

IV.2 Typologie selon le niveau du formalisme [Uschold et Gruninger, 1996] ;

IV.3 Typologie selon le niveau de complétude [Mizoguchi et al., 1998] ;

IV.4 Typologie selon le niveau de détail [Guarino, 1997].

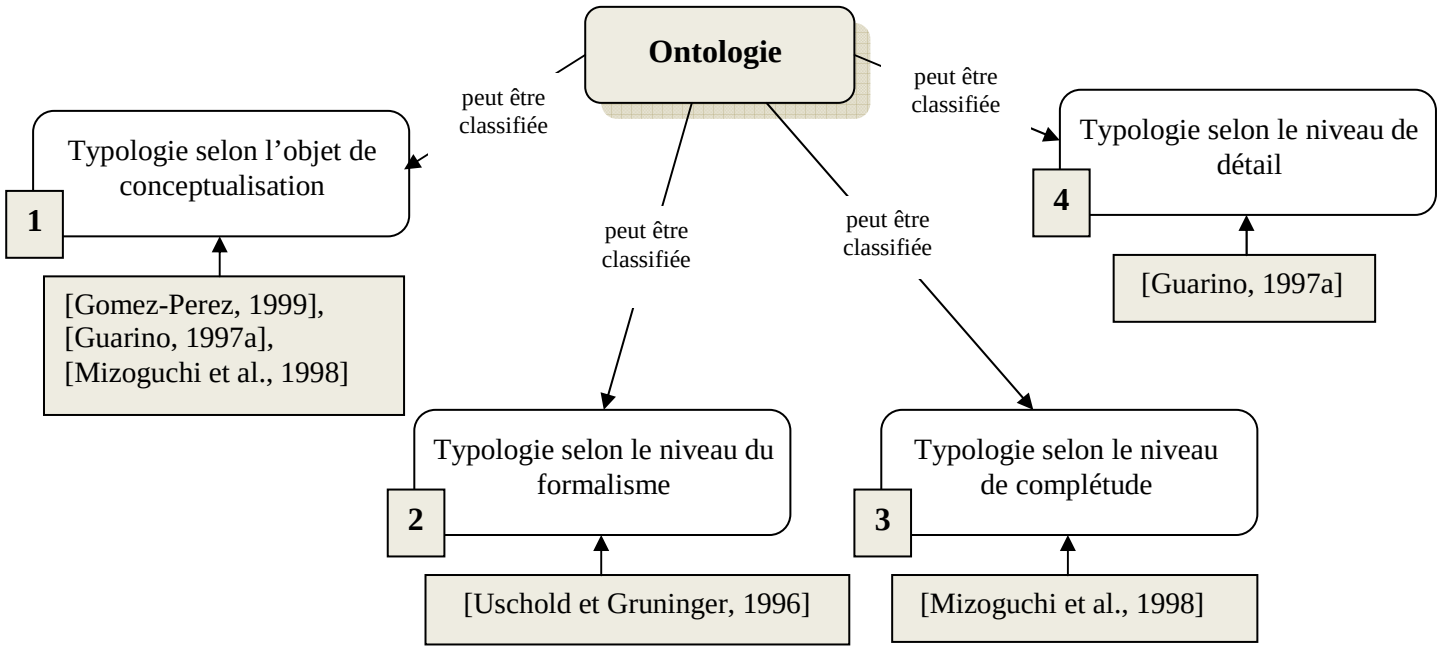


Figure 1: Typologies d'ontologies selon quatre dimensions de classification

IV.1 Dimension selon l'objet de conceptualisation :

IV.1.1 Ontologie de représentation [Méta ontologie] :

Ces ontologies conceptualisent les primitives des langages de représentation des connaissances. Par exemple, une ontologie sur le formalisme des Topic Maps comportera les concepts : Topic, Type de Topic, Association, Occurrence, Type Occurrence, etc.

IV.1.2 Ontologie supérieure ou de haut niveau :

Ces ontologies expriment des conceptualisations valables dans différents domaines. Ce type d'ontologies est fondé sur la théorie de l'identité, la méréologie (theory of whole and parts role) et la théorie de la dépendance. Son sujet est l'étude des catégories des choses qui existent dans le monde. Comme les concepts de haute abstraction tels que les entités, les évènements, les états, les processus, les actions, le temps, l'espace, les relations, les propriétés, etc.

IV.1.3 Ontologies applicatives :

Ces ontologies contiennent des connaissances du domaine nécessaires une application donnée, elles sont spécifiques et non réutilisables. Par exemple, dans le contexte du e-Learning, une application peut être : la Formation de Statistiques et Probabilités.

IV.1.4 Ontologie de domaine :

Cette ontologie régit un ensemble de vocabulaire et de concept qui décrit un domaine d'application ou de monde cible, Elle permet de créer des modèles d'objets du monde cible. L'ontologie du domaine est une méta-description d'une représentation des connaissances, c'est à dire une sorte de méta-modèle de connaissance dont les concepts et les propriétés sont de type déclaratif. La plupart des ontologies existantes sont des ontologies de domaine.

IV.1.4.1 Caractéristiques d'une ontologie de domaine :

- Formelle: Une ontologie est une conceptualisation basée sur une sémantique formelle.
- Consensuelle: Une ontologie est une conceptualisation acceptée par une communauté qui peut être plus ou moins large.
- Référençable: Chaque concept d'une ontologie est associé à un identifiant permettant de le référencer à partir de n'importe quel environnement, indépendamment de l'ontologie dans laquelle il a été défini.

IV.2 Dimension selon le formalisme :

Propose une classification comprenant quatre catégories :

a) Informelle : l'ontologie est exprimée en langue naturelle. Cela peut permettre de rendre plus compréhensible l'ontologie pour l'utilisateur, mais cela peut rendre plus difficile la vérification de l'absence de redondances ou de contradictions ;

b) Semi informelle : l'ontologie est exprimée dans une forme restreinte et structurée de la langue naturelle ; cela permet d'augmenter la clarté de l'ontologie tout en réduisant l'ambiguïté ;

c) Semi formelle l'ontologie est exprimée dans un langage artificiel défini formellement;

d) Formelle : l'ontologie est exprimée dans un langage artificiel disposant d'une sémantique formelle, permettant de prouver des propriétés de cette ontologie.

L'intérêt d'une ontologie formelle est la possibilité d'effectuer des vérifications sur l'ontologie: complétude, non-redondance, consistance, cohérence, etc. Uschold et Gruninger (1996) expliquent que :

"*le degré de formalisation exigé du langage pour l'ontologie dépend étroitement du degré d'automatisation dans les diverses tâches impliquant l'ontologie. Si une ontologie est une aide à la communication entre personnes, alors la représentation de l'ontologie peut être informelle du moment qu'elle est précise et qu'elle capture les intuitions de chacun. Cependant, si l'ontologie*

doit être employée par des outils logiciels ou des agents intelligents, alors la sémantique de l'ontologie doit être rendue beaucoup plus précise". [Uschold et Grüninger, 1996]

IV.3 Typologie selon le niveau de complétude :

Selon [Bachimont, 2000], on peut définir trois niveaux de complétude :

IV.3. 1 Niveau 1 : Sémantique :

Tous les concepts, caractérisés par un terme/libellé, doivent respecter les quatre principes différentiels :

- Communauté avec l'ancêtre;
- Différence, spécification, par rapport l'ancêtre;
- Communauté avec les concepts frères, situés au même niveau;
- Différence par rapport aux concepts frères.

Ces principes correspondent l'engagement sémantique qui assure que chaque concept aura un sens univoque et non contextuel associé. Deux concepts sémantiques sont identiques si l'interprétation du terme/libellé à travers les quatre principes différentiels aboutit un sens équivalent.

IV.3.2 Niveau 2 : Référentiel *:*

Les concepts référentiels ou formels, se caractérisent par un terme/libellé dont la sémantique est définie par une extension d'objets. L'*engagement ontologique* spécifie les objets du domaine qui peuvent être associés au concept, conformément à sa signification formelle.

Deux concepts formels seront identiques s'ils possèdent la même extension. Par exemple, les concepts d'*étoile du matin* et d'*étoile du soir* associés à Vénus.

IV.3.3 Niveau 3 : Opérationnel *:*

Les concepts du niveau opérationnel sont caractérisés par les opérations qu'il est possible de leur appliquer pour générer des inférences ou *engagement computationnel*. Deux concepts opérationnels sont identiques s'ils possèdent le même potentiel d'inférence.

IV.4 Typologie selon le niveau de détail :

On peut distinguer les ontologies selon le niveau de description utilisé [Psyché et al., 2003] :

a) Granularité fine *:*

Ce niveau correspond à des ontologies très détaillées, elles possèdent ainsi un vocabulaire plus riche capable d'assurer une description détaillée des concepts pertinents d'un domaine ou d'une tâche. Ce niveau de granularité peut s'avérer utile lorsqu' il s'agit d'établir un consensus entre les agents qui l'utiliseront ;

b) Granularité large *:*

Ce niveau correspond à des vocabulaires moins détaillés. Par exemple, les scénarios d'utilisation spécifiques où les utilisateurs sont déjà préalablement d'accord à propos d'une conceptualisation sous-jacente [Fürst, 2002], [Guarino, 1997]. Les ontologies de haut niveau possèdent une granularité large, compte tenu que les concepts qu'elles traduisent sont normalement raffinés subséquemment dans d'autres ontologies de domaine ou d'application [Fürst, 2002].

V. Processus de construction d'une ontologie :

L'analyse des travaux actuels permet de dégager un certain consensus sur le processus de construction d'une ontologie.

La construction d'une ontologie est une collaboration qui fait réunir des experts du domaine de connaissance, des ingénieurs de la connaissance, voir les futurs utilisateurs de l'ontologie. Cette collaboration ne peut être fructueuse que si les objectifs des processus ont été clairement définis, ainsi que les besoins qui en découlent.

Il y a trois étapes :

V.1. Conceptualisation

V.2. Ontologisation

V.3. Opérationnalisation

V.1 Conceptualisation :

La conceptualisation permet d'aboutir à un modèle informel donc sémantiquement ambiguë et généralement exprimé en langage naturel. Cette étape permet de dégager les concepts et les relations sur ces concepts à partir de données bruts. Elle est réalisée par un expert du domaine assisté de l'ingénieur de connaissance et aboutit un modèle conceptuel.

La problématique de la conceptualisation est essentiellement de reconnaître, dans les données brutes les connaissances qui relèvent du domaine considéré, de les recenser

de façon exhaustive. Les étapes suivantes du processus vont conduire progressivement la formalisation de ses connaissances.

V.2 Ontologisation :

La première étape de cette formalisation est l'Ontologisation, qui consiste en une formalisation partielle, sans perte d'information, du modèle conceptuel obtenu dans l'étape précédente. En fait cette étape produit un résultat en deux parties :

1. Une partie formelle : dispose d'une sémantique précise, ou du moins consensuelle.
2. Une partie informelle : qui ne dispose pas de sémantique claire ou consensuelle, exprimé dans un langage naturel ou semi-structuré.

V.3 Opérationnalisation :

Consiste à formaliser complètement l'ontologie obtenue précédemment dans le cadre d'un langage de représentation de connaissance formel et opérationnel. On obtient alors une représentation formelle des connaissances de domaine.

VI. Outils de construction d'ontologies:

Ces outils ont été proposés pour aider à la conception manuelle d'ontologies. Ces outils permettent d'éditer une ontologie, d'ajouter des concepts et des relations, etc. Ils intègrent différents langages de formalisation (RDF, OWL). Certains doivent être installés en local alors que d'autres sont distribués sur le Web. Ces outils sont décrits plus spécifiquement.

VI.1 DOE : DOE (Differential Ontologie Editor) [Troncy et Issac ,2002] [DOE, 2002] offre la possibilité de construire les hiérarchies de concepts et relations en utilisant les principes différentiels énoncés par B. Bachimont [Bachimont, 2000], puis en ajoutant les concepts référentiels. La sémantique des relations est ensuite précisée par des contraintes. Ce n'est qu'une fois l'ontologie ainsi structurée qu'elle est formalisée en utilisant la syntaxe XML.

VI.2 PROTEGE-2000 : Parmi les outils non liés à des formalismes de représentation, citons l'outil PROTEGE-2000 [Noy et al., 2000], [Protégé, 2000]. PROTEGE-2000 est une interface modulaire permettant l'édition, la visualisation, le contrôle (vérification des contraintes) d'ontologies, et la fusion semi-automatique d'ontologies à l'aide du plugin Prompt [Noy et Musen, 2000]. Le modèle de connaissances sous-jacent à PROTEGE-2000 est issu du modèle de frames et contient des classes (concepts), des slots (propriétés) et des facettes (valeurs des propriétés et contraintes), ainsi que des instances de classes et des propriétés. Il autorise la définition de méta-classes, dont les instances sont des classes, ce qui permet de créer son propre modèle de connaissances avant de bâtir une ontologie.

Il présente une interface graphique très agréable et simple à utiliser. Son utilisation dans le cadre de notre étude a été bénéfique pour pouvoir manipuler les ontologies.

VI.3 OntoEdit : OntoEdit (Ontology Editor) [OntoEdit,2004] est également un environnement de construction d'ontologies indépendant de tout formalisme. Il permet l'édition des hiérarchies de concepts et de relations et l'expression d'axiomes algébriques portant sur les relations, et de propriétés telles que la généricité d'un concept. Des outils graphiques dédiés à la visualisation d'ontologies sont inclus dans l'environnement.

OntoEdit intègre un serveur destiné à l'édition d'une ontologie par plusieurs utilisateurs. Un contrôle de la cohérence de l'ontologie est assuré à travers la gestion des ordres d'édition. Enfin, un plug-in nommé ONTOKICK offre la possibilité de générer les spécifications de l'ontologie par l'intermédiaire de questions de compétences. OntoEdit gère de nombreux formats de représentation de connaissances dont OWL, RDFS et FLogic.

VII. Langages de construction d'ontologies :

Dans cette partie, nous présentons différents langages pour créer des ontologies. Ces langages ont été conçus pour répondre notamment aux besoins du web sémantique, qui a fait son apparition dans le World Wide Web (WWW). A l'inverse du Web d'aujourd'hui, qui est d'abord conçu pour l'interprétation et l'utilisation humaine, le Web Sémantique est une vision d'un Web qui serait non-ambiguë et compréhensible par une machine. Le Web sémantique reste entièrement fondé sur le Web "classique" et ne le remet pas en cause.

Cela reste un moyen de publier et consulter des documents, mais les documents traités par le Web sémantique. Ceci est réalisé par l'annotation du contenu du Web, par la description de propriétés et de relations. Cette annotation est réalisée avec un langage raisonnablement expressif, possédant une sémantique bien définie et permettant le partage ainsi que la réutilisation des données au travers de différentes applications. Le consortium W3C a défini un cadre général pour ce type de descriptions basé sur RDF (Ressource Description Framework) pour la représentation de connaissance et OWL (Ontology Web Language) pour la modélisation sémantique de connaissances.

VII.1. eXtended Markup Language et XML Schema

L'eXtended Markup Language[W3Cd,2004] (XML) est un langage de description et d'échange de documents structurés, issu de SGML (Standard Generalized Markup Language) et défini par le consortium Web. XML permet de décrire la structure arborescente de documents à l'aide d'un système de balises permettant de marquer les éléments qui composent la structure et les relations entre ces éléments. XML ne pose aucune contrainte sémantique sur la description

des informations, il ne constitue donc pas un langage de modélisation d'ontologies à lui seul. Avec XML nous pouvons décrire une Personne Toto âgée de 37 ans qui habite 12 rue des pins. Pour un agent l'exemple ci-dessous correspond est compris comme une simple arborescence de chaînes de caractères.

```
<Personne id='Toto'>
<Adresse>12 rue des pins</Adresse>
<Age>37</Age>
</Personne>
```

XML Schema [W3Ce,2004] (XML-S) est un outil de définition de grammaires caractérisant des arborescences de documents (notion de validité syntaxique). Avec les schémas XML, il est possible de contraindre la structure arborescente d'un document mais pas la sémantique des informations contenues dans ce document.

VII.2 Resource Description Framework et RDF Schema

Le Ressource Description Framework [W3Cb,2004] (RDF) est un modèle pour la représentation de méta-données à propos de ressources. Cette représentation est faite sous la forme d'un triplet <sujet, prédicat, objet> telle que:

– Sujet : la ressource que l'on définit

– Prédicat : la propriété de la ressource, qui est une liaison étiquetée et orientée du sujet vers l'objet.

– Objet : la valeur de la propriété pouvant être une autre ressource ou bien un littéral.

RDF est à la base exprimé sous la forme d'un graphe orienté mais on peut le décrire par la syntaxe XML. On parle alors de RDF/XML qui par abus de langage peut être appelé RDF. En prenant l'exemple défini pour XML, nous pouvons définir avec RDF (figure 2) un graphe (figure 3) que l'agent comprendra comme un concept "Toto" qui possède les propriétés adresse et âge.

Si RDF fournit une capacité d'échange de connaissances, il ne permet pas à l'utilisateur de définir le vocabulaire des termes à utiliser, ni d'établir la sémantique des objets utilisés.

```
<rdf :RDF>
<rdf :Description about='Toto'>
<rdf :Property about='adresse'>
12 rue des pins
</rdf :Property>
<rdf :Property about='age'>
37
</rdf :Property>
</rdf :Description>
</rdf :RDF>
```

Figure 2: Représentation RDF/XML de Toto

37 ans qui habite au 12 rue des Pins

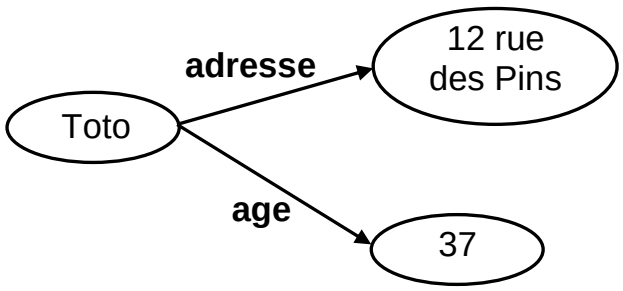


Figure 3 : Représentation graphique de

"**Toto**" **37 ans qui habite au 12 rue des pins**

RDF Schema[W3Cc,2004] ou RDFS est un langage permettant de définir des propriétés sémantiques pour les ressources par un schéma. Dans un schéma on peut définir de nouvelles ressources comme des spécialisations d'autres ressources. Les schémas contraignent aussi le contexte d'utilisation des ressources. Avec RDFS de nouvelles notions sémantiques apparaissent. La principale est la distinction entre une classe (concept d'une ontologie) et une instance (individu d'une ontologie). Voici d'autres notions définies dans RDFS :

– La notion de classe (rdfs :Class) qui peut être rapprochée de la notion de concept d'une ontologie.

– La notion de sous-classe (rdfs :subClassOf) qui est une spécialisation d'une classe déjà définie.

– La notion de type (rdf :Type) : les instances d'une classe propriété et sous propriété (prédicat ou rôle de l'ontologie)

– Les notions de source (rdfs :domain) et de cible (rdfs :range) d'une propriété.

La hiérarchie de subsomption ou taxinomie peut être décrite en utilisant la propriété rdfs: subClassOf. Cette hiérarchie définit les relations de spécialisation d'une classe Ressource par rapport à une classe Objet. Cette relation s'applique également aux relations par la propriété rdfs: SubPropertyOf.

Soit P une propriété de rdfs: range R et de rdfs: domain D, alors une propriété S est une rdfs: SubPropertyOf de P si le rdfs: range de S est R ou une sous-classe de R et le rdfs: domain de S est D ou une sous-classe de D.

Sur l'exemple de "Toto", avec le schéma RDF nous définissons le concept de Personne (figure2 et figure 3) avec les types des objets de chaque propriété, une taxinomie de concepts ou hiérarchie de subsomption (figure 5) ainsi que l'instance Toto (figure 6). Nous avons vu que RDF et RDFS permettent de définir sous la forme d'un graphe de triplets des données ou des méta-données. Cependant, il est impossible de raisonner sur ces représentations car la sémantique (hors subsomption) reste très limitée. C'est ce manque de sémantique que OWL comble par l'apport d'un vocabulaire plus riche.

```
<rdf :RDF>
<rdfs :Class rdf :about='Animal'>
<rdfs :subClassOf rdf :resource='Thing'/>
</rdfs :Class>
<rdfs :Class rdf :about='Personne'>
<rdfs :subClassOf rdf :resource='Animal'/>
</rdfs :Class>
<rdf :Property about='age'>
<rdfs :domain rdf :resource='Animal'/>
<rdfs :range rdf :resource='xsd :integer'/>
</rdf :Property>
<rdf :Property about='adresse'>
<rdfs :domain rdf :resource='Personne'/>
<rdfs :range rdf :resource='xsd :string'/>
</rdf :Property>
</rdf :RDF>
```

Figure 4: Représentation RDFS du concept Personne

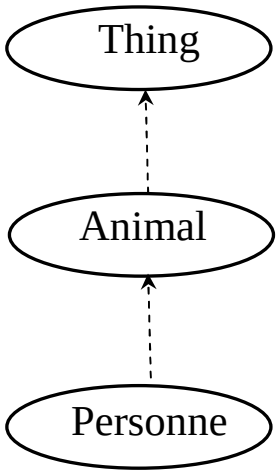


Figure 5: Taxinomie des concepts

```
<Personne rdf :ID='Toto'>
<age rdf :resource='37' />
<adresse rdf :resource='12 rue des pins' />
</Personne>
```

Figure 6 : Représentation RDFS de Toto

VII.3 Ontology Web Language

OWL (Ontology Web Language [W3Ca,2004], [Zhihong et Mingtian, 2003]) est un langage fondé sur la syntaxe RDF/XML et héritier des travaux de DAML+OIL [Schneider et al., 2001].

OWL n'est pas simplement une extension de RDF, il introduit l'aspect sémantique lui manquant, comme les outils de comparaison de propriétés et de classes (identité, équivalence, contraire, cardinalité, symétrie, etc). Ainsi par ses primitives plus riche, OWL offre une capacité d'interprétation plus grande au machine que RDF et RDFS.

OWL se décompose en trois sous-langages OWL Lite, OWL DL et OWL Full, offrant des capacités d'expression croissantes et donc destinés à des différentes utilisations. OWL Lite est le sous langage de OWL le plus simple car son utilisation est contrainte.

Voici la liste des constructeurs proposés par OWL Lite :

Catégories	**Constructeurs**
RDF Schema	Class, rdfs :subClassOf, rdf :Property, rdfs :subPropertyOf, rdfs :domain, rdfs :range, Individual
(In)Equalité	equivalentClass, equivalentProperty, sameAs, differentFrom, AllDifferent, distinctMembers
Restrictions	onProperty, allValuesFrom, someValuesFrom
Cardinalités (0 ou 1)	minCardinality, maxCardinality, cardinality
Intersection	intersectionOf
Propriétés	SymmetricProperty,FunctionalProperty, ObjectProperty, DatatypeProperty, inverseOf, TransitiveProperty, InverseFunctionalProperty

Table 1 : La liste des constructeurs proposés par OWL Lite

Nous illustrons ces constructeurs par un exemple d'ontologie décrite en OWL Lite

Catégories	Exemple
RDF Schema	Personne : subClassOf (Thing) Femme : subClassOf (Personne) Enfant : subClassOf (Personne)
(In)Equalité	Fille : equivalentClass (intersectionOf(Femme, Enfant))
Intersection	Parent : intersectionOf (Personne,
Restrictions	Restriction (
Cardinalités	minCardinality(1), onProperty (aEnfant)))
Propriétés	aParent : ObjectProperty (Enfant, Personne) aEnfant : inverseOf (aParent)

Table 2: Les constructeurs OWL Lite par un exemple d'ontologie décrite en OWL Lite

OWL DL est plus complexe que OWL Lite, il est fondé sur la logique de description *SHIN(D)* (d'ou son nom OWL Description Logics). Malgré sa complexité OWL DL garantit la complétude des raisonnements (calculabilité des inférences) et leur décidabilité (leur calcul se fait en une durée finie).Voici la liste des constructeurs ajoutés par OWL DL par rapport OWL Lite :

Catégories	Exemple
Axiome de Classe	oneOf (enumeration), dataRange, disjointWith
Expressions booléennes	unionOf, complementOf
Cardinalités (0,n)	minCardinality, maxCardinality, cardinality
Individu cible d'une propriété	hasValue

Table 3: La liste des constructeurs ajoutés par OWL DL par rapport OWL Lite

Nous illustrons ces constructeurs par un exemple d'ontologie décrite en OWL DL

Catégories	**Exemple**
Axiome de Classe	Gender : oneOf(Male, Female)
Expressions booléennes	Tante : intersectionOf (Femme , unionOf (aNeuve, aNiece))
Individu cible d'une propriété	Homme : intersectionOf (Personne, hasValue(sexe,Male))

Table 4: Les constructeurs OWL Lite par un exemple d'ontologie décrite en OWL DL

OWL Full est la version la plus complexe d'OWL, mais également celle qui permet le plus haut niveau d'expressivité. Son utilisation n'est contrainte que par le langage RDF, mais elle ne garantit pas la complétude et la décidabilité des calculs liés à l'ontologie. La syntaxe ne subit pas de modification par rapport à OWL DL mais OWL Full permet une utilisation sans contrainte sur l'utilisation des constructeurs.

Exemple: utiliser un concept à la place d'un individu Adulte : IntersectionOf(Personne, hasValue(age,Majorite)), Majorité étant un concept remplaçant l'individu de type Age et de valeur 18.

Les 3 niveaux d'OWL présentent une hiérarchie sur la validité des ontologies :

– une ontologie OWL Lite valide est également une ontologie OWL DL valide

– une ontologie OWL DL valide est également une ontologie OWL Full valide

VIII. Conclusion :

Dans ce chapitre nous avons discuté les ontologies, largement utilisée dans notre travail, et leurs utilisations en ingénierie des connaissances. Les ontologies servent à décrire formellement des concepts et des relations entre concepts. Les concepts décrivent un système donné et participent à la signification des termes. Pour matérialiser ces concepts et relations, l'ontologie doit être construite et exprimée dans un langage.

Les apports de l'utilisation des ontologies sont divers. Les ontologies jouent un rôle important dans les systèmes à base de connaissance. Outre la réutilisation et le partage de connaissances, elles permettent de faciliter la communication entre les acteurs de différentes organisations. Elles permettent, en particulier, la réalisation de l'interopérabilité entre différents systèmes.

Nous avons ensuite, présenté les différentes classifications ou typologies des ontologies, qui peuvent être classées selon plusieurs dimensions (selon l'objet de conceptualisation, le niveau de complétude, le niveau du formalisme, et selon le niveau de détail).

Construire une ontologie est une collaboration qui fait réunir des experts du domaine de connaissance, des ingénieurs de la connaissance, voir les futurs utilisateurs de l'ontologie. Cette collaboration ne peut être fructueuse que si les objectifs des processus ont été clairement définis, ainsi que les besoins qui en découlent. Conformément aux étapes de construction, la conceptualisation vient en premier, Elle est suivie de l'étape d'Ontologisation, et enfin, de l'étape d'Opérationnalisation.

Nous avons ensuite, présenté une série d'outils, qui ont été proposés pour aider à la création et la manipulation d'ontologies, Parmi eux, PROTEGE-2000 et DOE (Differential Ontologie Editor) et OntoEdit (Ontology Editor), puis, Nous avons intéressé aux différents Langages de construction d'ontologies tels que le XML et XML Schema, RDF et RDF Schema, et OWL.

Dans le chapitre suivant, nous nous intéressons à l'alignement des ontologies qui constitue une notion importante dans notre travail.

Chapitre 2 :
Alignement des Ontologies

Introduction :

De plus en plus d'applications industrielles accessibles sur le Web ou par d'autres réseaux, offrent des services fondés sur le partage et l'interopérabilité des structures de connaissances issues de diverses organisations, et représentant divers aspects d'un domaine modélisé. Plusieurs approches existent pour mettre en œuvre cette interopérabilité. Parmi ces approches, on distingue l'alignement d'ontologies, qui consiste à identifier des relations entre les éléments de différentes ontologies. Euzenat et ses collègues [Euzenat et al., 2004] identifient différents types de méthodes d'alignement d'ontologies: terminologiques, structurelles, extensionnelles et sémantiques, qui proviennent de disciplines variées, telles que la fouille de données, les sciences du langage, les statistiques ou la représentation des connaissances.

Dans les systèmes pair à pair, l'hétérogénéité des données gérées et partagées par les différents pairs constituent un réel problème pour une exploitation optimale des ressources fournies. L'alignement des ressources dans le contexte pair à pair est un axe de recherche extrêmement important.

Dans ce chapitre nous allons commencer par une définition de l'alignement des ontologies, puis nous présentons les différentes applications possibles de ce nouveau domaine pour étayer l'intérêt accordé à l'alignement des ontologies, ensuite, nous allons examiner les techniques et les méthodes utilisées dans la littérature qui attaquent le problème de recherche de la similarité, de la dissimilarité ou de la correspondance entre deux entités en général, qu'elles apparaissent dans des schémas, ou dans des ontologies représentées soit en RDF(S), soit en OWL.

Ensuite, des approches qui emploient ces techniques seront présentées. Enfin, un tableau récapitulatif résumera une vue globale des approches et leurs relations avec des techniques présentées.

I. Définition de l'Alignement des Ontologies:

L'alignement des ontologies permet d'établir des liens sémantiques entre les concepts et des relations inter-ontologies, comme l'illustre la définition de Namyoun: "*Ontology alignment is the task of creating links between two original ontologies. Ontology alignment is made if the sources become consistent with each other but are kept separate. Ontology alignment is made when they usually have complementary domains*». [Namyoun et al., 2006]

Donc l'alignement d'ontologies est un processus de découverte des correspondances entre deux ontologies sources, dont l'entrée est constituée d'un ensemble d'ontologies et la sortie, formée des correspondances entre ces ontologies (voir la figure7).

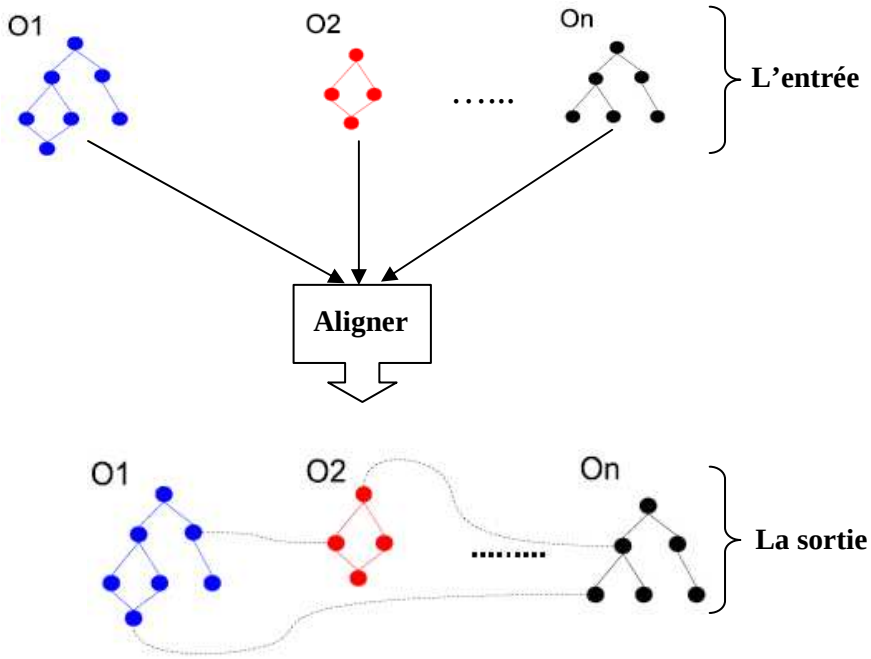


Figure 7: le processus d'alignement des ontologies

Nous présentons tout d'abord une distinction entre les termes souvent utilisés et liés à la problématique de l'alignement d'ontologie à savoir : les mappings, le matching, le matcher, l'alignement, la fusion et l'intégration d'ontologies.

- **Correspondances ou Mappings :** Les mappings sont des relations entre les éléments de deux représentations (ontologies, schémas de bases de données, etc.), indiquant une similarité relative selon une mesure donnée [Klein et al., 2001].

- **Appariement ou Matching :** Le matching d'ontologies est le processus de définition d'un ensemble de fonctions permettant de spécifier des « correspondances » entre termes. [Xu et al., 2003], [He et al., 2003].
- **Les méthodes de comparaison ou matchers :** Un matcher est une fonction utilisée pour calculer la distance entre deux entités. Les matchers sont des fonctions qui peuvent être combinées dans le processus de matching.
- **Alignement d'ontologies:** L'alignement d'ontologies est le processus d'établissement de liens de correspondances entre deux ontologies. Il est appliqué si les ontologies concernées deviennent homogènes entre elles et ceci tout en les gardant séparées (pas de fusion d'ontologies). Cette catégorie de mapping d'ontologies est faite habituellement quand les ontologies sources appartiennent à des domaines complémentaires.
- **Fusion d'ontologies:** La fusion d'ontologies est le processus de création d'une seule ontologie rassemblant les connaissances de deux ou plusieurs ontologies existantes et différentes qui décrivent le même sujet ou appartiennent au même domaine d'application. L'ontologie générée inclut les informations de toutes les ontologies sources, mais est plus ou moins inchangée.
- **Intégration d'ontologies :** L'intégration d'ontologies est un processus de construction d'une nouvelle ontologie qui n'est pas forcément destinée à remplacer les autres (ces dernières peuvent continuer à être utilisées par ailleurs, à être mises à jour, à évoluer, etc.). Ces différentes ontologies peuvent être connexes.

II. Quelques domaines d'application de l'alignement des ontologies :

II.1. L'ingénierie ontologique :

Est un contexte où les concepteurs d'ontologies sont confrontés à l'hétérogénéité de ces dernières, plus précisément, par rapport aux applications suivantes :

La construction d'ontologies: ces dernières années, le maître mot dans la démarche de construction des ontologies est la réutilisation d'ontologies déjà existantes, car la construction d'ontologies à partir de zéro est un processus long, couteux et très laborieux, parallèlement, elle accentue le phénomène de l'hétérogénéité des ontologies, multipliant le nombre d'ontologies décrivant le même domaine (surtout lorsqu'on sait que l'objectif ultime du web sémantique est d'arriver à instaurer une ontologie de référence pour chaque domaine). Dans ce contexte, l'alignement des ontologies est la solution pour réaliser l'intégration et le rapprochement de ces différentes structures.

L'évolution des ontologies: Beaucoup d'ontologies sont en continuelle évolution comme la Geneontology[1], et de ce fait, plusieurs versions de la même ontologie sont disponibles, mettant les développeurs et les ingénieurs de la connaissance dans la confusion, ne sachant pas ce qui a changé, l'alignement va permettre d'identifier les différences entre deux versions : les entités qui ont été ajoutés, supprimés ou renommés.

II.2. L'intégration d'information :

C'est une application classique de l'alignement d'ontologies, elle comprend l'intégration des schémas, les entrepôts de données, l'intégration des données et l'intégration des catalogues.

La figure 8 présente un scénario général de l'intégration d'information, étant donné, un ensemble de sources locales d'information (ontologies locales LO_1, …,LO_n) qui stockent leurs informations dans des formats différents (SQL, XML, RDF), fournit aux utilisateurs une interface de requêtes uniforme à travers une ontologie globale ou commune (CO) à toutes les sources d'information. Ce qui permet aux utilisateurs d'adresser des requêtes directement à l'ontologie globale.

Les sources de données sont transformés en ontologies locales qui sont alignées par rapport à une ontologie globale, les alignements obtenus aident à générer les médiateurs qui, à leurs tours, transforment les requêtes adressées à l'ontologie globale en requêtes pour les sources d'information locales et traduisent les réponses dans l'autre sens.

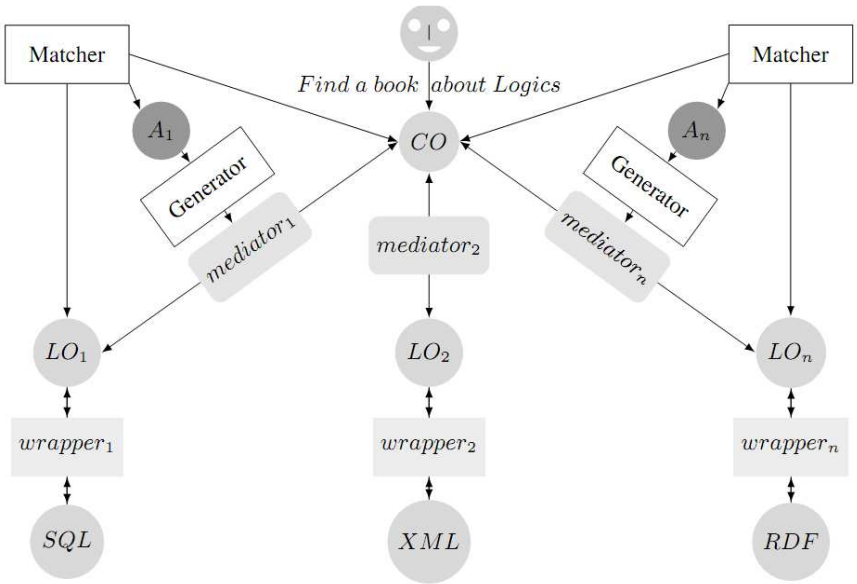


Figure 8: Scénario centralisé d'intégration d'information [Euzenat et Shvaiko, 2007]

[1] http://www.geneontology.org

II.3. Le Partage des informations dans les systèmes pair à pair (P2P) :

Le système Pair à pair est un modèle de communication distribué dans lequel les pairs ont des capacités fonctionnelles équivalentes dans les échanges de données et de services [Zaihrayeu, 2006].

Les pairs doivent s'échanger des informations alors qu'ils utilisent des terminologies différentes (le problème de traduction des requêtes et de leurs réponses). Une des étapes pour résoudre ce problème serait d'identifier les relations qui existent entre leurs ontologies respectives, autrement dit, réaliser un alignement d'ontologies.

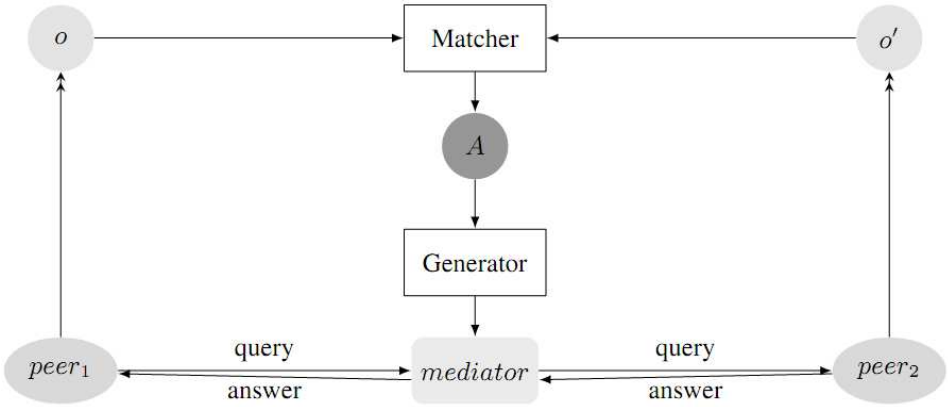


Figure 9 : Réponse à une requête dans un système P2P

[Euzenat et shvaiko, 2007]

II.4. La Composition des services web :

Les services web sont des processus qui exposent leurs interfaces aux utilisateurs du web qui les invoquent. Les services web sémantiques fournissent un moyen plus riche et plus précis de décrire les services à travers les langages de représentation des connaissances et des ontologies [Fensel et al., 2007].

Par exemple, un service web fournit la description de son output à l'aide d'une ontologie et un autre service web utilise une seconde ontologie pour décrire son input, aligner ces deux ontologies permettrait de vérifier si ce qui a été délivré par le premier service correspond à ce qui était attendu par le second service et cela grâce à un médiateur entre ces deux services, généré à partir de l'alignement des deux ontologies précédemment citées.

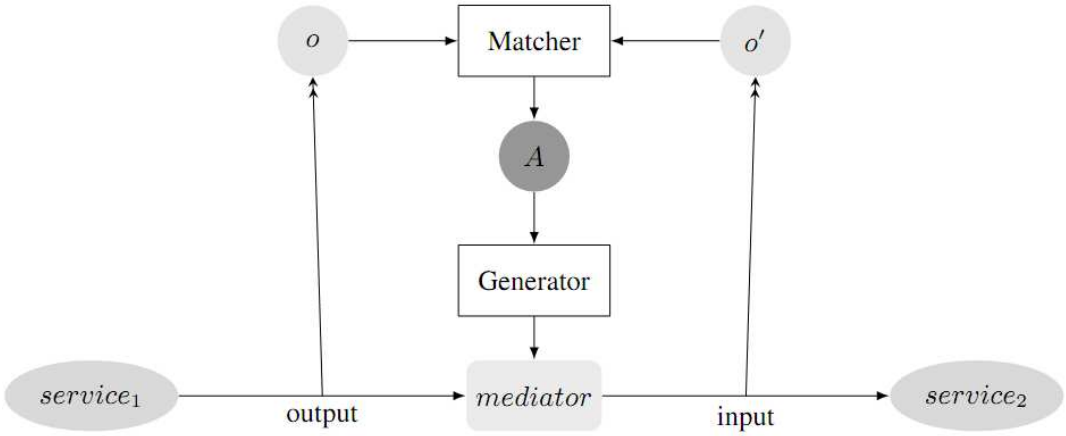


Figure 10 : Composition d'un service web

[Euzenat et shvaiko, 2007]

II.5. La communication entre agents :

Lorsque deux agents autonomes et conçus indépendamment se rencontrent, ils ont la possibilité de s'échanger des messages mais peu de chances pour se comprendre s'ils ne partagent pas le même langage et la même ontologie. L'alignement de leurs ontologies respectives intervient à ce niveau pour traduire les messages ou bien intégrer des passerelles entre leurs axiomes dans le modèle propre à chaque agent (pour pouvoir interpréter les messages).

III. Les méthodes d'Alignement d'Ontologies :

L'objectif de l'alignement est d'identifier les relations entre les entités de différentes ontologies. Ces relations sont découvertes à travers les mesures de similarité entre ces entités.

Dans cette partie, nous allons examiner les techniques et les méthodes utilisées dans la littérature qui attaquent le problème de recherche de la similarité, de la dissimilarité ou de la correspondance entre deux entités en général, qu'elles apparaissent dans des schémas, ou dans des ontologies représentées soit en RDF(S), soit en OWL.

Ensuite, des approches qui emploient ces techniques seront présentées. Enfin, un tableau récapitulatif résumera une vue globale des approches et leurs relations avec des techniques présentées.

III.1 Les méthodes de base pour mesurer la similarité :

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

III.1.1 La similarité :

Dans la littérature, plusieurs travaux sur la mesure de similarité sémantique entre les objets d'une ontologie ont été développés dans différents contextes. La similarité sémantique est une évaluation du lien sémantique entre deux concepts dont le but est d'estimer le degré par

lequel les concepts sont proches dans leur sens [Resnik, 1999]. La définition donnée par Lin [Lin, 1998] de la similarité sémantique repose sur trois suppositions. La similarité entre deux concepts est liée aux caractéristiques qu'ils ont en commun (plus ils ont de caractéristiques communes, plus les concepts sont similaires) et à leurs différences (plus deux concepts sont différents, moins ils sont similaires). La similarité maximale est obtenue lorsque deux concepts sont identiques.

Dans notre contexte, la notion de similarité sémantique, qui est également appelée la proximité sémantique. est la quantité qui reflète la force du rapport entre deux objets ou deux caractéristiques.

Concrètement, ceci peut être réalisé par exemple en définissant une similarité topologique, ou en employant des ontologies pour définir une distance entre les termes (une métrique naïve pour des termes représentés comme noeuds dans une hiérarchie ontologique pourrait être la distance minimale des arcs différents entre les deux noeuds), ou en employant des moyens statistiques pour faire corréler des mots et des contextes textuels.

Dans la plupart des approches dans notre contexte, la notion de similarité sémantique est vue comme celle de la similarité topologique en mathématiques, où on l'associe à une fonction, appelée fonction de la similarité. La définition de cette fonction de la similarité peut changer selon les approches, selon les propriétés souhaitées. La valeur de cette fonction est souvent comprise entre 0 et 1, ce qui permet des possibilités d'interprétation probabiliste de la similarité. Des propriétés ou des caractéristiques communes possibles de la fonction sont des caractéristiques positives, auto similaires ou maximales, symétriques ou réflexives. On peut aussi trouver d'autres caractéristiques telles que la finitude ou la transitivité.

Définition 1 (Similarité) : La similarité S: O×O → R est une fonction d'une paire d'entités à un nombre réel exprimant la similarité entre ces deux entités telle que:

- $\forall a, b \in O, S(a, b) \geq 0$ (positivité)
- $\forall a, b, c \in O, S(a, a) \geq S(b, c)$ et $S(a, a) = S(a, b) \Leftrightarrow a = b$ (autosimilarité ou maximalité)
- $\forall a, b \in O, S(a, b) = S(b, a)$ (symétrie)
- $\forall a, b, c \in O, S(a, b) = S(b, c) \Rightarrow S(a, b) = S(a, c)$ (transitivité)
- $\forall a, b \in O, S(a, b) \leq \infty$ (finitude)

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

Définition 2 (Dissimilarité): La dissimilarité DS: O×O → R est une fonction d'une paire d'entités à un nombre réel exprimant la dissimilarité entre ces deux entités telle que:

- $\forall a, b \in O, DS(a, b) \geq 0$ (positivité)
- $\forall a, b, c \in O, DS(a, a) \leq DS(b, c)$ et $DS(a, a) = 0$ (minimalité)
- $\forall a, b \in O, DS(a, b) = DS(b, a)$ (symétrie)
- $\forall a, b \in O, DS(a, b) \leq \infty$ (finitude)

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

Définition 3 (Distance) : La distance $D: O \times O \rightarrow R$ est une fonction de la dissimilarité satisfaisant la définitivité et l'inégalité triangulaire :

- $\forall a, b \in O, D(a, b) = 0 \Leftrightarrow a = b$ (définitivité)
- $\forall a, b, c \in O, D(a, b) + D(b, c) \geq D(a, c)$ (inégalité triangulaire)

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

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

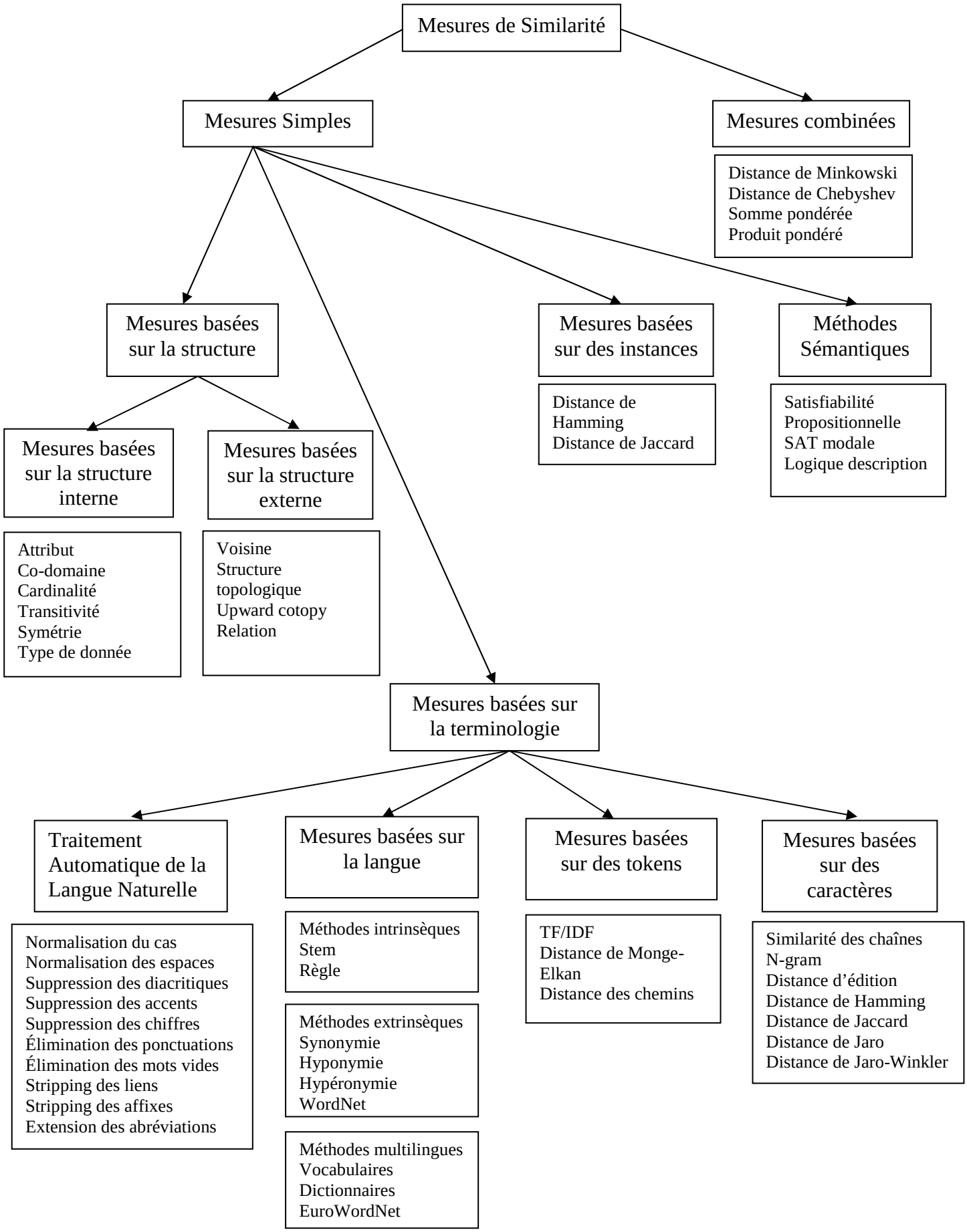


Figure 11 : Les catégories des mesures de similarité selon différentes techniques

[Bach Than Le, 2006]

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

La Figure 11 résume différentes mesures de la similarité, catégorisée selon les techniques utilisées. Ce résumé est une synthèse des travaux présentés dans [Rahm et Bernstein, 2001], [Euzenat et al., 2004] et [Euzenat et shvaiko, 2007].

III.1.2 Les méthodes terminologiques :

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

III.1.2.1 Les méthodes se basent sur des chaînes de caractères :

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

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

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

caractères CR) trouvées dans une chaîne de caractères par un seul caractère d'espace. Par exemple, l'expression «ma voiture » est normalisée à «ma voiture».

- *Suppression des signes diacritiques ou des accents (aigus, graves…) :* ce type de traitement remplace des caractères avec des signes diacritiques par caractères correspondants sans signes diacritiques. Par exemple, le mot « Hanoï » est remplacé par le mot « Hanoi » sans changer la signification du mot (Hanoï est la capitale du Vietnam). Cependant, certaines suppressions changeront la signification du terme : « là » (adverbe de lieu) et « la » (article).
- *Suppression des chiffres.*
- *Élimination des ponctuations.*
- *Élimination des mots vides* (les mots contenant peu d'informations tels que «est», « un », «les»…)
- *Suppression des affixes* (préfixes, suffixes).
- *Extension des abréviations.*
- *Tokenisation.*
- *Lemmatisation* (passer au singulier, à l'infinitif pour les verbes, au masculin pour les adjectifs…)

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

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

Définition 5 (Similarité de Jaccard) : Soit s et t deux chaînes de caractères.

Soit S et T les ensembles des caractères de s et t respectivement. La similarité de Jaccard est une fonction de la similarité $S_{Jaccard}$: S x $S \rightarrow [0, 1]$ telle que :

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

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

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

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

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

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

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

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

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

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

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

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

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

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

III.1.2.2 Les distances basées sur des tokens :

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

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

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

Une mesure qui est largement employée dans le domaine de la recherche d'information, semble convenable ici. Il s'agit du TF/IDF (Term frequency/Inverse Document Frequency).

Dans sa conception originale, TF/IDF est employé pour mesurer la pertinence d'un terme dans l'ensemble de documents. La fréquence de terme, TF, dans un document donné montre l'importance de ce terme dans le document en question. La fréquence inverse de document, IDF, est une mesure de l'importance générale du terme dans l'ensemble de documents.

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

$$TF = \frac{n(t)}{N} \;\; et \;\; TF/IDF = TF * log\left[\frac{|D|}{d(t)}\right]$$

La similarité des entités textuelles peut être construite comme la valeur cosinus de deux vecteurs représentant ces entités, où chaque dimension correspond à un terme et sa valeur correspond à la valeur TF/IDF de ce terme. [Monge et Elkan, 1996] propose une méthode hybride pour comparer des chaînes de caractères longues, qui découpe ces deux chaînes en plusieurs chaînes plus courtes.

Ensuite, ces dernières sont comparées par une mesure de distance quelconque citée ci dessus. Enfin, les résultats obtenus sont combinés.

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

S x $S \rightarrow$ [0, 1] telle que :

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

III.1.2.3 Les méthodes linguistiques:

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

III.1.2.3.1 Les méthodes intrinsèques :

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

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

Table 5: Variantes du terme «enzyme activity» (extrait de [Euzenat et al., 2004])

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

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

Une approche plus complexe pour déterminer le radical exact d'un mot est la lemmatisation. Ce processus comprend la détermination de la partie du discours (catégorie lexicologique) d'un mot, et l'application des règles de normalisation différentes pour chaque partie du discours. Cette approche exige la connaissance de la

grammaire d'une langue, des règles différentes… Elle est donc lourde, compliquée et difficile à implémenter.

Le premier stemmer publié a été écrit par Julie Beth Lovins [Lovins, 1968]. Ensuite un autre stemmer a été développé par Martin Porter [Porter, 1980]. Ce dernier est très largement utilisé, et est devenu l'algorithme standard utilisé pour chercher des radicaux dans la langue anglaise. L'algorithme est implémenté dans plusieurs langages de programmation. Snowball[1], un framework pour implémenter des algorithmes de stemmers, leurs variantes ou bien des algorithmes de stemmers pour des autres langues (par exemple le finnois, le russe, le danois, l'allemand, le français…), est aussi créé par Porter.

III.1.2.3.2 Les méthodes extrinsèques :

Les méthodes extrinsèques calculent la valeur de similarité entre deux termes en employant des ressources externes telles que des dictionnaires, des lexiques ou des vocabulaires. La similarité est décidée grâce aux liens sémantiques déjà existants dans ces ressources externes tels que des liens synonymes (pour l'équivalence), des liens hyponymes/ hypernymes (pour la subsomption).

Par exemple, à l'aide des ressources des synonymes, « voiture » et « bagnole » sont dites similaires. Typiquement, WordNet[2], un système lexicologique, est employé pour trouver des relations telles que la synonymie entre des termes, ou pour calculer la distance sémantique entre ces termes, en utilisant des liens sémantiques dans WordNet, afin de décider s'il existe une relation entre eux.

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

1 http://snowball.tartarus.org/

2 http://wordnet.princeton.edu/

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

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

III.1.3 Les méthodes structurelles :

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

III.1.3.1 Les méthodes structurelles internes :

Ces méthodes comparent les structures internes des entités (*exemple*, intervalle de valeur, cardinalité d'attributs, etc.). Dans la plupart des cas, ce sont des informations concernant des attributs de l'entité, telles que des informations du co-domaine des attributs, celles de la cardinalité des attributs, celles des caractéristiques des attributs (la transitivité, la symétrie), ou celles des autres types de restriction sur des attributs. Par exemple, en considérant l'entité : le concept « Humain », nous pouvons exploiter des informations concernant des attributs de ce concept tels que l'intervalle des valeurs de donnée pour l'attribut « hasAge », à savoir [0, 150] ; la cardinalité de l'attribut « hasSpouse », à savoir 1 ; ou bien la caractéristique transitive de l'attribut « hasAncestor ».

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

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

III.1.3.2 Les méthodes structurelles externes :

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

- Leurs super-entités directes (ou toutes leurs super-entités) sont similaires.
- Leurs sœurs (ou toutes leurs sœurs, qui sont les entités ayant la même super-entité directe avec les entités en question) sont déjà similaires.
- Leurs sous-entités directes (ou toutes leurs sous-entités) sont déjà similaires.
- Leurs descendants (entités dans le sous-arbre ayant pour racine l'entité en question) sont déjà similaires.
- Toutes (ou presque toutes) leurs feuilles (les entités de même type, qui n'ont aucune sous-entité, dans le sous-arbre ayant pour racine l'entité en question) sont déjà similaires.
- Toutes (ou presque toutes) les entités dans les chemins de la racine aux entités en question sont déjà similaires.

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

Des mesures ont été proposées pour comparer deux entités basées sur des structures taxonomiques. [Valtchev et Euzenat, 1997] proposent une mesure récursive de la dissimilarité topologique structurelle qui se base sur la distance du chemin le plus court dans un graphe. [Mädche et Staab, 2002] introduit la notion de « upward cotopy » qui est

définie UC(c, H) = {c'∈H ; c ≤ c'}, l'ensemble d'entités qui sont super-entités de l'entité c dans la hiérarchie H. La mesure de similarité, inspirée de la distance de Jaccard (voir Définition 5), est alors définie ainsi:

Définition 11 (Similarité de « upward cotopy ») : Soit H_1 et H_2 deux hiérarchies.
Soit e et e' deux entités dans H_1 et H_2 respectivement. La similarité de «upward cotopy» entre e et e' est une fonction de la similarité S_{cotopy} : O x O → [0, 1] telle que :

$$\overline{S_{cotopy}}(e, e') = |\frac{UC(e, H_1) \cap UC(e', H_2)}{UC(e, H_1) \cup UC(e', H_2)}|$$

Une autre approche pour déduire la similarité entre deux entités exploite des relations reliant ces deux entités. L'idée est que si l'on a deux entités similaires A et A', et si elles sont connectées par un même type de relation R avec deux autres entités B et B', alors on peut déduire que B et B' sont d'une façon ou d'une autre similaires. De même, si on sait que A et A' sont similaires, B et B' sont aussi similaires, alors les relations de A-B et de A'-B' peuvent être similaires. L'idée peut être étendue pour un ensemble d'entités et de relations : si on a un ensemble de relations $R_1 \ldots R_n$ qui sont similaires avec un autre ensemble de relations $R'_1 \ldots R'_n$, alors les entités qui sont les domaines (ou les co-domaines) de ces relations sont considérées comme similaires.

Cependant, cette approche pose un autre problème : comment peut-on dire que deux relations sont similaires ? Cette approche est basée sur la similarité des relations pour impliquer la similarité de leurs entités de domaine ou leurs entités de co-domaine.

Des relations entre les entités peuvent être considérées comme d'autres entités, elles peuvent être aussi organisées dans une hiérarchie de relations, et donc, le calcul de la similarité entre les relations est également un grand problème.

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

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

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

III.1.4 Les méthodes extensionelles :

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

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

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

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

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

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

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

Une autre mesure qui partage l'idée similaire avec la mesure hybride (Définition 10) est définie dans [Valtchev, 1999] comme la similarité basée sur des correspondances (match-based similarity). Ici, la similarité entre deux ensembles est la similarité moyenne des paires des éléments dans Pairing, où Pairing est l'ensemble de correspondances ayant la somme maximale des toutes les similarités des paires dans l'ensemble. Le calcul de l'ensemble Pairing(S, T) est un problème d'optimisation qui maximise le total des similarités des paires des éléments de S et T.

Définition 15 (Similarité basée sur des correspondances) : Soit S et T deux ensembles d'entités. Soit S_q(s, t) une mesure de similarité quelconque de deux entités s et t. Soit Pairing(S, T)

l'ensemble de correspondances entre S et T ayant la somme maximale des valeurs de similarité de ces correspondances. La similarité basée sur des correspondances entre deux ensembles S et T est une fonction de la similarité $\mathcal{S}_{Corr}$: 2E x 2E → [0, 1] telle que :

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

Une autre mesure pour calculer la similarité entre deux ensembles emploie une technique d'analyse multidimensionnelle dans le domaine de la statistique [Cox, 1994].

L'hypothèse de cette technique est que si deux entités ont les distances très similaires à toutes autres entités, elles doivent être très similaires. Les entités dans des ensembles sont représentées par des vecteurs, dont la valeur d'une dimension est la similarité de l'entité en question avec une autre entité dans les deux ensembles. La similarité entre deux ensembles est donc la similarité (la valeur du cosinus) de deux vecteurs moyens de ces deux ensembles.

Définition 16 (Similarité des ensembles) : Soit S = {s_1, s_2,…} et T = {t_1, t_2,…} deux ensembles d'entités. Soit $S_q(s_i, t_j)$ une mesure de similarité quelconque de deux entités *s* et *t*.

Soit $\vec{s_i}$= *(sim(e_i,e_1), sim(e_i,e_2),… sim(e_i,f_1), sim(e_i,f_2),…)* le vecteur représentant de l'entité e_i. La similarité des ensembles entre S et T est une fonction de la similarité $\mathcal{S}_{Set}$: 2E x 2E→[0, 1] telle que:

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

III.1.5 Les méthodes sémantiques :

Les méthodes sémantiques comparent les interprétations (ou plus exactement les modèles) des entités, donc ces méthodes se basent sur :

- **Des modèles de logique:** tels que les techniques de la satisfiabilité propositionnelle (SAT) et les techniques fondées sur les logiques de descriptions (ou la SAT modale) ;
- et sur **des méthodes de déduction :** pour déduire la similarité entre deux entités.

Les approches dans [Giunchiglia et Shvaiko, 2003; Giunchiglia et al., 2004; Bouquet et al., 2003] emploient des techniques de satisfiabilité propositionnelle (SAT), qui permettent de retranscrire les problèmes d'appariement d'arbres –présentés sous la forme de deux hiérarchies de concepts et d'un ensemble de correspondances- en logique propositionnelle pour vérifier la validité d'un ensemble de ces formules qui est construit en traduisant des relations déjà connues et des relations à vérifier entre des entités vers des formules propositionnelles.

L'approche dans [Giunchiglia et Shvaiko, 2003] étend les méthodes proposées ci-dessus, qui sont pour le modèle de la SAT propositionnelles, vers le modèle de la SAT modale, qui peut aussi contenir des prédicats binaires. La limite du premier modèle est qu'il n'accepte que des

prédicats unaires qui sont des entités comme des concepts, des classes. Le dernier permet de calculer en plus avec des prédicats binaires tels que des relations, des attributs ou des propriétés (slots) et d'employer des opérateurs de la logique modale. La validité de l'ensemble de formules formulées en logique modale est aussi vérifiée en utilisant des procédures de recherche de la satisfiabilité (SAT). Si la validité est satisfaite, les relations hypothétiques entre des entités, qui sont des traductions de la requête sur la relation entre ces entités en logique modale, sont confirmées.

Les techniques des logiques de description [Baader et al.,2003] qui sont capables d'appliquer les algorithmes de subsomption inclus dans certains moteurs d'inférences pour déduire des correspondances entre les concepts d'ontologies, a partir de leurs représentations formelles, ces techniques peuvent être employées pour vérifier des relations sémantiques entre des entités telles que :

- L'équivalence (la similarité est égale à 1),
- la subsomption (la similarité est de 0 à 1) ou
- l'exclusion (la similarité est égale à 0), et permettent donc de déduire la similarité de deux entités.

III.2 Les méthodes de combinaison des similarités:

Une entité peut être considérée sous plusieurs différents aspects, soit en s'appuyant sur son nom, soit sur ses attributs, ou soit sur ses relations avec d'autres entités. La similarité entre deux entités peut donc être calculée en se basant sur plusieurs aspects. Sur chaque aspect, les caractéristiques d'une entité sont comparées avec les caractéristiques correspondantes d'une autre par une des mesures de similarité de base présentées dans la section III.1, cela retourne une valeur de la similarité (ou de la dissimilarité/distance). Il faut donc un moyen pour combiner toutes les valeurs de similarité calculées de chaque aspect pour produire une seule valeur de similarité représentative pour deux entités à comparer. Cette partie analyse quelques approches existantes dans la littérature.

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

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

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

Cette distance est une mesure généralisée avec différentes valeurs de p, p >= 1. Quand p est égale à 1, elle devient la distance de « city block » et quand p = 2 elle devient la distance euclidienne. La distance de Chebyshev (appelée aussi la distance de valeur maximum) est un cas spécial de la distance de Minkowski avec p = ∞ :

$$DS_{Chebyhev}(x, y) = max_i \ DS(x_i, y_i)$$

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

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

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

En général, la somme des poids est égale à 1: $\sum_{i=1}^{n} w_i = 1$, dans ce cas, nous avons la version normalisée de DS_{sp}. Une autre mesure analogue à la somme pondérée est le produit pondéré. Cependant, un inconvénient de cette mesure est que le résultat sera égal à 0 si une des dimensions est égale à 0.

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

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

Les approches d'alignement des schémas ou des ontologies présentées dans la section suivante emploient une ou plusieurs mesures de similarité présentées dans la section III.1 pour calculer les valeurs de similarité entre des entités, ensuite retourner des alignements entre deux schémas ou deux ontologies en évaluant ces valeurs de similarité.

Toutes les approches, qui combinent des valeurs de similarité calculées par différentes mesures, emploient la méthode de la somme pondérée (Définition 18). Cependant, certaines

approches (par exemple Anchor-PROMPT) déduisent des alignements en examinant des critères heuristiques sans utiliser des méthodes de combinaison des similarités.

IV. Quelques approches d'alignement d'ontologies:

Dans cette partie, nous allons analyser des approches déjà existantes dans la littérature qui concernent le problème d'alignement d'ontologies.

Anchor-PROMPT (Stanford Medical Informatics SMI)

Anchor-PROMPT est un algorithme qui a été développé par Noy et Musen [Noy et Musen, 2001] afin de découvrir automatiquement les termes sémantiquement similaires. Anchor-PROMPT traite une ontologie comme un graphe, les nœuds de ce graphe représentant les classes et les arcs représentant les propriétés. L'algorithme utilise deux paires de termes relatifs comme entrée. Il analyse les chemins dans le sous-graphe délimité par des ancres et détermine quelles sont les classes qui apparaissent fréquemment dans des positions similaires sur des chemins similaires.

L'algorithme cherche alors des termes le long des chemins qui pourraient être similaires aux termes d'autres chemins.

Ces nouveaux termes relatifs sont identifiés par une similarité qui peut être modifiée pendant l'évaluation d'autres chemins dans lesquels ces termes apparaissent. Les termes qui sont fortement semblables sont présentés à l'utilisateur pour améliorer l'ensemble des suggestions possibles.

Anchor-PROMPT ne cherche que des correspondances des concepts, pas des correspondances des relations. En outre, il emploie des noms de relation pour des étiquettes sur les arcs et la comparaison des chaînes de caractères de ces étiquettes n'est que la comparaison simple. Ainsi si les noms de relation sont différemment définis, l'algorithme ne fonctionnera pas bien. Les résultats retournés par l'algorithme seront également limités si les structures des ontologies sont différentes (par exemple l'une est profonde avec beaucoup de concepts au milieu, et l'autre est peu profonde). L'algorithme rencontre des problèmes si une hiérarchie a seulement quelques niveaux et si la plupart des relations sont associées aux concepts au-dessus de la hiérarchie.

Cupid (Microsoft research)

Madhavan et ses collègues [Madhavan et al., 2001] ont proposé une approche de recherche des correspondances combinant un module sophistiqué de mise en correspondance des noms et un algorithme de mise en correspondance au niveau structurel (Approche hybride). L'algorithme se compose de trois étapes :

a) le niveau linguistique : les valeurs de similarité entre les noms des éléments (les étiquettes) sont calculées en employant des techniques et mesures linguistiques (III.1.2) telles que la normalisation des chaînes des caractères (III.1.2.1), la catégorisation, les mesures de similarité sur des préfixes, des suffixes, l'emploi le thésaurus des synonymes, des hypernymes;

b) le niveau structurel : La similarité structurelle de deux éléments est la similarité de deux arbres dont leurs racines sont les éléments à comparer. Cette dernière similarité est calculée en se basant principalement sur la similarité des feuilles de ces arbres.

c) la valeur de similarité finale de deux éléments est la somme pondérée de deux valeurs calculées dans deux étapes précédentes. Si cette valeur finale est plus grande qu'un seuil prédéfini, deux éléments sont considérés similaires. (voir la figure suivante)

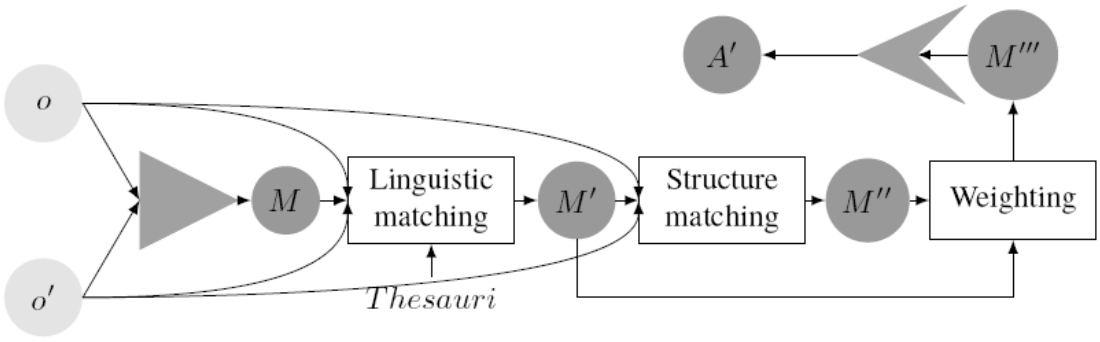


Figure 12 : l'architecture de Cupid [Euzenat et Shvaiko, 2007]

GLUE (Université de Washington)

GLUE [Doan et al., 2003] est un système qui emploie une approche de type " machine learning " (Une technique d'apprentissage telle que Naïve Bayes) pour créer des mappings semi-automatiques entre ontologies hétérogènes. Il y a plusieurs modules d'apprentissage (learners), qui sont entraînés par des instances des ontologies, donc Il se base sur des données d'instances, une ontologie étant vue comme une taxonomie de concepts. GLUE se focalise sur la détermination des mapping de type 1-à-1. La similarité de deux concepts A et B dans deux taxonomies *O*1 et *O*2 utilise l'ensemble des instances des deux concepts qui convergent.

Pour déterminer si une instance du concept B est également une instance du concept A, un classifieur est d'abord construite en utilisant les instances de A comme ensemble d'apprentissage.

Ce classifieur est ensuite utilisé à son tour pour traiter les instances de B. Le classifieur décide alors pour chaque instance de B, s'il est également une instance de A ou non. Avec ces classifications, quatre probabilités sont calculées : $P(A,B)$, $P(\bar{A},B)$, $P(A,\bar{B})$ et $P(\bar{A},\bar{B})$. La probabilité $P(A,\bar{B})$ par exemple, s'interprète par l'appartenance de l'instance du domaine à A et la non appartenance à B. Ces quatre cas peuvent ensuite être employées comme paramètres pour calculer la distribution de probabilité commune pour les concepts A et B, laquelle est une fonction écrite par l'utilisateur [Kalfoglou et Schorlemmer, 2003]. Un inconvénient de cette approche est qu'elle se fonde principalement sur les instances des ontologies, qui ne sont pas toujours abondamment disponibles pour plusieurs ontologies. Un autre inconvénient est que l'ontologie est modélisée comme une taxonomie des concepts et que chaque concept a quelques attributs. Avec cette organisation, GLUE n'emploie pas des informations contenues dans la taxonomie (hiérarchie) des relations. GLUE fait également l'utilisation modeste des informations sur la taxonomie des concepts.

COMA et COMA++ (Université de Leipzig)

Do et Rahm ont développé **COMA** (COmbination of MAtching algorithms) [Do et Rahm, 2002] comme un système permettant de mettre en correspondance des schémas (des bases des données, de XML) automatiquement ou bien manuellement. Donc c'est un système générique interactif permettant de trouver des appariements entres schémas. Il fournit une bibliothèque extensible des algorithmes de mise en correspondance de base (appelés matchers) et quelques mécanismes pour combiner des résultats des algorithmes de base afin d'obtenir une valeur de similarité finale de deux éléments dans deux schémas. La bibliothèque des matchers se compose de :

a) 6 matchers simples qui emploient des techniques linguistiques (III.1.2) telles que la similarité des préfixes, des suffixes, n-gram, la distance d'édition (Définition 6), la similarité phonétique (soundex), la synonymie avec des dictionnaires externes, la similarité des types de données prédéfinie ;
b) 5 matchers hybrides qui combinent des matchers simples précédents en exploitant quelques informations structurelles telles que des chemins entre des éléments, la similarité de leurs enfants.

COMA présente quelques stratégies pour agréger des résultats de similarité pour chaque paire d'éléments, calculés de différents matchers tels que le choix de la valeur maximale, la

valeur moyenne, la somme pondérée ou la valeur minimale. La sélection de bonnes correspondances parmi toutes les paires d'éléments repose sur une des stratégies suivantes : la paire ayant la valeur de similarité agrégée maximale, ou les paires ayant les valeurs de similarité agrégées dépassant un seuil. Le système COMA permet aussi d'exécuter plusieurs itérations de calcul et d'utiliser le résultat de l'étape précédente dans le calcul de similarité de l'étape actuelle, ou d'avoir des interactions des utilisateurs dans chaque itération.

COMA + + est la version évoluée de COMA , il fournit une mise en œuvre plus efficace que l' algorithme COMA et une interface graphique plus riche pour l'utilisateur.

S-Match (Université de Trento)

S-Match est une approche pour le matching des hiérarchies de classifications [Giunchiglia et al., 2004]. Les auteurs mettent en œuvre un opérateur de correspondance qui admet en entrée deux structures sous forme de graphes (par exemple, schémas de base de données ou ontologies) et produit un mapping entre les éléments qui sont en correspondance sémantique, cette approche est basée sur l'idée d'employer le moteur de la satisfiabilité propositionnelle (SAT) [Giunchiglia et Shvaiko, 2003] pour le problème de mise en correspondance des schémas. Il prend comme entrée deux graphes des concepts (schémas), et produit en sortie des rapports entre les concepts tels que l'équivalence, overlapping, différence (mismatch), plus général ou plus spécifique. L'idée principale de cette approche est d'utiliser la logique pour coder le concept d'un nœud dans le graphe et d'appliquer SAT pour trouver des rapports.

Le concept à un nœud, qui est alors transformé en formule propositionnelle, est la conjonction de tous les concepts des étiquettes des nœuds sur le chemin de la racine du graphe jusqu'au nœud en question. Le concept d'étiquette d'un nœud est construit en deux étapes :

(i) la normalisation de l'étiquette : telle que la tokenization, la lemmatisation.

(ii) l'extraction du sens de l'étiquette normalisée (des lemmes) à partir de WordNet [Miller, 1995]. Ensuite, les relations sémantiques (l'équivalence, plus général, plus spécifique) entre deux étiquettes de deux schémas sont :

  (i) calculées grâce aux « matchers », les modules qui calculent la similarité entre deux étiquettes en employant des mesures de similarité de base (III.1.2) telles que la similarité des préfixes, des suffixes, la distance d'édition (Définition 6), la similarité de n-gram ; ou bien

  (ii) déduites en employant des matchers qui exploitent la sémantique dans WordNet, la similarité entre des hiérarchies, la similarité entre des commentaires [Giunchiglia et Yatskevich, 2004]. Ces relations sémantiques sont aussi encodées en

logique. Enfin, le rapport entre deux concepts qui doit être prouvé est également converti en formule propositionnelle. Le moteur SAT calcule sur l'ensemble de formules propositionnelles pour vérifier si le rapport supposé est vrai ou faux. Cela permet donc de déduire des correspondances entre deux ontologies.

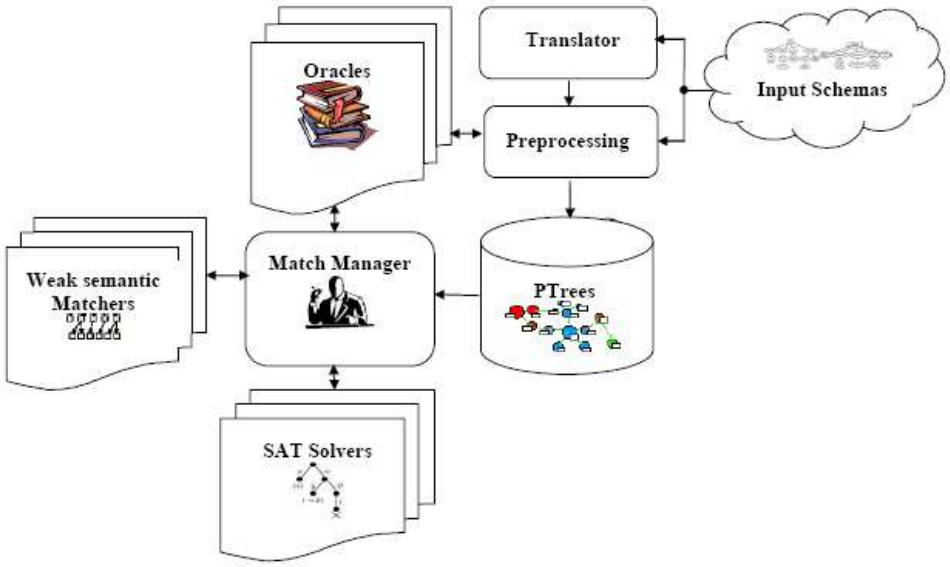


Figure 13: Architecture de la plate forme S-match

QOM (Université de Karlsruhe)

QOM (*Quick Ontology Mapping*) est un algorithme qui a été conçu pour fournir un outil efficace de matching pour la création au vol des alignements entre les ontologies [Ehrig et Staab, 2004]. Afin d'accélérer l'identification des similarités entre deux ontologies, QOM ne compare pas celles de la première ontologie avec toutes les entités de la seconde ontologie, mais emploie des heuristiques (par exemple, labels semblables) pour abaisser le nombre de mappings candidats. QOM représente des ontologies en RDF(S) et utilise aussi des mesures de similarité telles que l'égalité des chaînes de caractères, la similarité des chaînes (Définition 6) sur des noms de concept, de relation, d'instance ; la similarité des ensembles (Définition 16)…

Plusieurs mesures de similarité sont calculées et servent d'entrée à une fonction d'agrégation. QOM applique une fonction qui fait ressortir différentes similarités élevées et basses. Les correspondances réelles entre les entités des ontologies sont extraites en appliquant un seuil de mesure agrégée de similarité. La sortie d'une itération peut être utilisée en tant qu'élément d'entrée pour l'itération suivante afin de raffiner le résultat. Après un certain nombre d'itérations, une table de correspondances entre les ontologies est obtenue. Ehrig et Staab ont

montré que leur QOM met en correspondance ces ontologies dans un délai acceptable sans sacrifier beaucoup la qualité du résultat final.

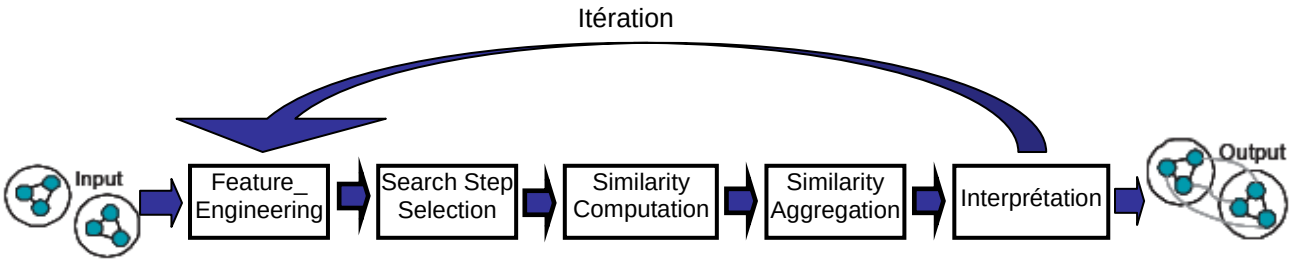


Figure 14: processus de QOM

NOM (Université de Karlsruhe)

NOM (Naïve Ontology Mapping) [Ehrig et Sure, 2004] est employé pour mettre en correspondance des ontologies (plutôt des thesaurus) telles que la hiérarchie des sujets de l'ACM, la structure des répertoires dans des ordinateurs personnels, le WordNet, ou l'UMLS. Ce sont des taxonomies ayant un nombre énorme de concepts (>104 concepts), Comme QOM, NOM utilise les mêmes mesures de similarité. Cependant, QOM a quelques modifications en comparaison avec NOM pour réduire la complexité de calcul, donc le temps d'exécution, telles que des stratégies de sélection des candidats à comparer (par hasard, des candidats ayant des étiquettes à proximité dans la liste triée…) ; des ensembles à comparer sont aussi limités (par exemple, QOM ne compare que deux ensembles de concepts parentaux directs de deux instances, au lieu de tous les concepts ancestraux comme dans NOM).

OLA (INRIA Rhône -Alpes et Université de Montréal)

OLA (OWL Lite Aligner) [Euzenat et Valtchev, 2004] est une classe d'algorithmes d'alignement d'ontologies qui exploite toutes les caractéristiques possibles des ontologies représentées en OWL-Lite et qui inclut (leurs types : classe, relation ou instance, leurs rapports avec d'autres entités : sous-classe, domaine, co-domaine…), et de combiner les valeurs de similarités calculées pour chaque paire d'entités de manière homogène. La combinaison est la somme pondérée des valeurs de similarité de chaque caractéristique. Les poids sont associés suivant le type d'entité à comparer et ses caractéristiques. Ils sont mis dans une matrice des poids et sont prédéfinis avant l'exécution de l'algorithme. Les mesures de similarité de base employées dans l'algorithme sont l'égalité des chaînes des caractères pour des URIs des entités, des mesures de similarité des suffixes ou des chaînes des caractères (III.1.2.1) pour des étiquettes des entités, la similarité (l'égalité) des types des données. Pour la similarité entre deux ensembles, le cas très souvent rencontré dans OWL (par exemple, en comparant deux entités, l'algorithme exploite la

similarité de deux ensembles d'entités qui sont sous-entités des entités en question), il utilise la mesure de similarité basée sur des correspondances (Définition 15). À partir des valeurs de similarité calculées par des mesures de base, l'algorithme applique un calcul du point fixe, avec des itérations pour améliorer la similarité de deux entités. Quand il n'y a plus d'améliorations, des alignements entre deux ontologies sont générés. Les travaux présentés dans cette section sont résumés dans le Tableau suivant :

Approche	Entrée	Représ-entation Interne	Mesures terminologiques	Mesures Structurelles	Mesures Extensionnelles	Mesures sémantiques	Mesures combinées	Observation
Anchor-PROMPT [Noy et Musen, 2001]	Ontologies	RDF(S), Graphe RDF	égalité des chaînes de caractères pour les étiquettes des concepts et des relations					Comparaison des chemins (informations structurelles) entres des paires d'ancres
Cupid Madhavan et al., 2001	Schéma générique	Graphe	Normalisation, catégorisation pour les noms d'élément de schéma Similarité des préfixes, des suffixes Thésaurus des synonymes, des Hypernymes	Similarité des arbres, des feuilles			Somme pondérée	La valeur de similarité agrégée dépasse un seuil prédéfini
GLUE Doan et al, 2002	Ontologies + instances	Non précise	Techniques terminologiques : tokenisation, stem égalité des chaînes de				Méta-learner : somme pondérée	Apprentissage automatique (Bayes naïf) à partir des

			caractères pour des tokens					valeurs textuelles, des noms des instances
COMA Do et Rahme, 2002	Schéma: Bases des données relationnelles, XML	Graphes Acycliques orientés	Similarité des préfixes, des suffixe, de n-gram, des synonymes, des types de donnée, distance d'édition, soundex,	Similarité agrégée des enfants, des feuilles ou des chemins entre des éléments			La valeur maximale ou minimale, la somme pondérée ou la valeur moyenne	MaxN, MaxDelta, ou dépasse un seuil
S-Match Giunchiglia et al, 2004-2005	Schéma des bases des données, de XML, hiérarchies des concepts, ontologies	Graphe	Similarité des préfixes, des suffixe, de n-gram, la distance d'édition pour les étiquettes	Distance des hiérarchies		Encode le concept d'un noeud en logique propositionnelle. Formule un ensemble d'équations logiques. Vérifie par SAT		

NOM Ehrig et sure, 2004	ontologies	RDF(S)	égalité des chaînes de caractères pour les URIs des concepts, des relations ou des instances similarité des chaînes pour des étiquettes des concepts, des relations ou des instances	Similarité des ensembles pour des super-concepts/relations, des sous-concepts/relations, des voisins, des concepts parentaux des instances	Similarité des ensembles pour des instances des concepts, des instances des relations		Somme pondérée Fonction de Sigmoid Technique d'apprentissage (trois niveaux) avec des réseaux neurologiques	
QOM Ehrig et Staab, 2004	Ontologies	RDF(S)	Comme NOM	Comme NOM avec des modifications : limitations sur des voisines directes	Comme NOM		Comme NOM	
OLA Euzenat et valtchev, 2004	Ontologies en OWL	OL-Graphe	égalité des chaînes de caractères pour les URIs des entités				Somme pondérée	Calcul du point fixe (information structurelle)

			Similarité des chaînes des caractères pour des étiquettes (non précise) Similarité des types des données					

Table 6: Résumé des approches d'alignement [Bach Than Le, 2006]

V. Conclusion:

Nous avons tenté, à travers ce chapitre, de mettre la lumière sur les aspects les plus importants de l'alignement des ontologies, qui est un domaine très vaste et par conséquent difficile à cerner.

Nous avons commencé par un aperçu sur le processus d'alignement des ontologies, qui permet **la découverte des correspondances** entre deux **ontologies**, dont l'entrée est constituée d'un ensemble d'ontologies et la sortie, formée des correspondances entre ces ontologies. Ensuite, Nous avons présenté une distinction entre les termes souvent utilisés et liés à la problématique de l'alignement d'ontologie à savoir : les mappings, le matching, le matcher, l'alignement, la fusion et l'intégration d'ontologies ;

Puis nous avons souligné l'importance de ce domaine via les innombrables possibilités en matière d'application ;

Ensuite, nous avons examiné les techniques et les méthodes utilisées dans la littérature qui attaquent le problème de recherche de la similarité, de la dissimilarité ou de la correspondance entre deux entités en général, qu'elles apparaissent dans des schémas, ou dans des ontologies représentées soit en RDF(S), soit en OWL.

Ensuite, des approches qui emploient ces techniques ont été présentées. Enfin, et pour une vue globale, nous avons fourni un tableau comparatif des approches d'alignement d'ontologies et des schémas, qui résume une vue globale des approches et leurs relations avec des techniques présentées.

Nous avons présenté des travaux dans la littérature qui abordent le problème d'alignement d'ontologies. Nous avons montré que les schémas ou les taxonomies peuvent être considérés comme des ontologies simples, et donc que les travaux dans le domaine d'alignement des schémas peuvent être adaptés pour mettre en correspondance les ontologies.

Dans le chapitre suivant, nous nous intéressons à la technologie Pair à Pair qui constitue une notion importante dans notre travail.

Chapitre 3 :
Architecture pair à pair

Introduction :

Le paradigme pair à pair (peer to peer ou P2P) est devenu un des plus importants domaines de l'informatique actuelle, particulièrement dans l'axe de l'informatique distribuée et Internet. Cela est dû principalement à la nature du pair à pair qui comporte plusieurs avantages, comme l'auto-organisation, la tolérance aux pannes, l'autonomie, l'aspect dynamique …

En plus de ces caractéristiques, le pair à pair propose des solutions dans plusieurs domaines d'applications tels que le calcul distribué, le partage de fichiers, l'agrégation des ressources…, pour toutes ces raisons, le concept du pair à pair a suscité beaucoup d'intérêt dans la communauté scientifique et professionnelle.

Nous consacrons tout ce chapitre à la présentation de ce concept afin de préparer le contexte de notre travail : nous tentons dans la première section de donner quelques définitions précises de ce qu'est le pair à pair, Nous nous intéressons ensuite, dans la deuxième section aux différents caractéristiques et les objectifs du pair à pair, Dans la troisième section, nous essayons de présenter les différents types des systèmes pair à pair, a savoir le pair à pair structuré et le pair à pair non structuré. La quatrième section s'occupe de présenter quelques travaux d'alignement des ontologies dans un système pair à pair. Cette section énumère quelques travaux de référence dans ce contexte. Le chapitre se termine par une discussion sur les aspects traités dans ce chapitre.

I. Qu'est-ce qu'un réseau de pair à pair ?

Afin de clarifier certaines ambiguïtés ontologiques, nous donnons quelques définitions précises de ce qu'est le pair à pair:

- « On peut définir le pair à pair (Peer-To-Peer ou P2P) comme une classe d'applications qui prennent avantage des ressources de stockage, de traitement processeur, de contenu et de la présence humaine aux différents nœuds de l'Internet. Les opérations d'accès à ces ressources décentralisées se font dans un environnement où les connexions sont instables et non contrôlées. Cela implique qu'un système pair à pair doit être autonome et indépendant de tout serveur central. » [Shirky, 2000].

- « Le terme pair à pair est défini aussi, comme étant une classe d'applications et de systèmes, qui emploient les ressources locales dans un environnement décentralisé. » [Milojicic et al., 2002].

Ainsi,

- « Le pair à pair désigne tout système distribué dont les participants partagent des ressources locales. Ces dernières peuvent êtres des fichiers, des capacités de calculs, de la bande passante du réseau etc. En plus, ce partage doit être accessible directement à tous les participants, sans passer par un serveur dédié. Dans ce contexte, le paradigme pair à pair procure pour chaque participant du réseau, les capacités d'un client et d'un serveur en même temps (chaque ordinateur possède des capacités et des responsabilités équivalentes) ».

II. Caractéristiques et objectifs des systèmes Pair à Pair :

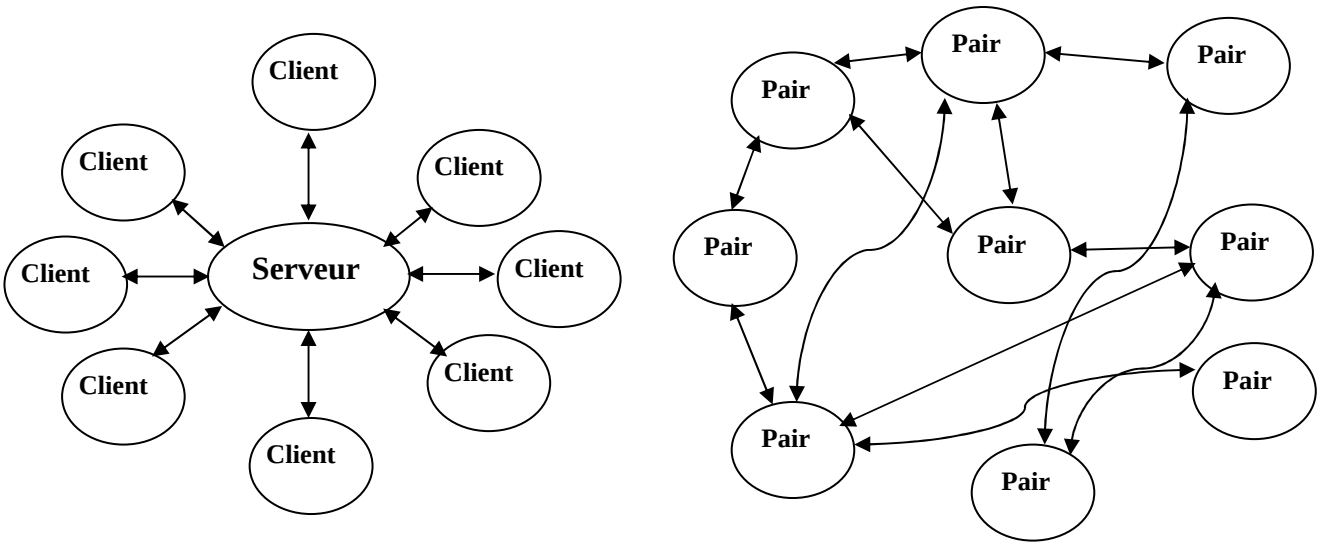
Après cette brève définition du terme pair à pair, nous présentons les caractéristiques et les objectifs les plus importants de ce paradigme.

II.1. Caractéristiques des systèmes pair à pair:

Le paradigme pair à pair possède plusieurs caractéristiques. Ces dernières sont principalement: la décentralisation, l'autonomie, la nature ad hoc, l'auto-organisation, la tolérance aux pannes, l'extensibilité et la montée en charge :

• **La décentralisation** est une des caractéristiques les plus importantes dans un système pair à pair. Dans le Client/Serveur, l'ensemble des ressources est fourni exclusivement par un serveur dédié. Or, dans le modèle pair à pair, chaque pair est égal aux autres. Il fournit des ressources comme un serveur et il établit des requêtes comme un client. La réalisation d'un tel système est néanmoins une tâche difficile, vu le grand dynamisme du système.

D'après la figure 15, on remarque que le modèle Client/Serveur concentre les services dans un seul nœud. Or, Dans le pair à pair, tous les nœuds peuvent êtres des clients et des serveurs en même temps.



(a) Client/Seveur **(b) Pair à Pair**

Figure 15 : Topologie réseaux du client/serveur et du Pair à Pair

• **L'autonomie** est une conséquence directe de la caractéristique précédente. Un pair a la possibilité de contrôler ses propres accès au système, ses ressources partagées et la façon de les partager.

• **La nature ad hoc** du pair signifie que chaque participant peut se connecter ou de se déconnecter sans aucun contrôle. En plus, les participants n'utilisent pas nécessairement les mêmes infrastructures réseaux. En fait, chaque groupe de pairs peut choisir entre diverses technologies, qui ne sont pas nécessairement compatibles. Cela pose un problème majeur pour assurer la communication et l'organisation du système entier. Afin de remédier à ce problème, l'auto organisation est un aspect très important qui doit être pris en considération dans les systèmes pair à pair.

• **L'auto-organisation** est définie comme le processus ou l'organisation d'un système est gérée de façon spontanée. Cela veut dire que l'organisation n'est pas contrôlée par un environnement ou un système externe. Comme les participants dans un système pair à pair ont les mêmes privilèges et le même degré d'autonomie, cette caractéristique a un intérêt primordial pour le bon fonctionnement du système. Une conséquence immédiate de l'auto organisation est d'assurer la capacité de la montée en charge et de l'extensibilité dans un système pair à pair.

• **La tolérance aux pannes** constitue une autre caractéristique du paradigme pair à pair, Dans un modèle Client/Serveur, la disponibilité des serveurs est vitale pour les services du réseau. Or, l'indisponibilité d'une partie des participants dans le pair à pair ne cause pas la défaillance de

tout le système. Dans ce cas, le système utilise sa caractéristique d'auto-organisation pour continuer à fournir les services aux participants existants.

• **L'extensibilité et la montée en charge** permettent d'avoir un nombre important de participants avec une évolution continue du système. Cela est assuré par des techniques et des protocoles de communication adaptés.

II.2. Objectifs des systèmes pair à pair :

Le paradigme pair à pair peut être utilisé pour satisfaire plusieurs besoins et peut être appliqué dans plusieurs domaines. Parmi ces derniers, on peut citer le calcul distribué, l'agrégation des ressources et le partage des fichiers.

• **Le calcul distribué** permet de partager les capacités du traitement processeur d'un pair. Afin de bénéficier de ce service, chaque projet doit décomposer le problème à traiter en plusieurs sous problèmes de taille minimale. Ensuite, chaque pair traite un problème à part, pour avoir un grand degré de distribution. Parmi les projets de référence dans ce domaine, on cite SETI@home [Seti, 2001]. Ce projet a pour objectif d'analyser les données d'un radio télescope pour la recherche d'une intelligence extraterrestre. SETI@home se base sur des ordinateurs connectés à Internet, dont la contribution des participants se résume à installer une application spéciale qui utilise la capacité du processeur local pour accomplir un calcul spécifique.

• **L'agrégation des ressources de stockage** permet d'avoir un système de gestion de fichier de taille importante. Chaque participant contribue avec une partie de ses capacités de stockage pour construire le système de stockage global. L'intérêt majeur de ce système est de minimiser les coûts en se basant sur des ressources déjà existantes. Mais, la disponibilité non assurée des participants constitue une des faiblesses de ce système.

• **Le partage des fichiers** constitue actuellement le service le plus sollicité dans un système pair à pair. Dans une application de partage de fichiers, chaque participant doit en premier lieu localiser les pairs qui ont le fichier voulu. Pour cela, un protocole de recherche adéquat est utilisé. Par la suite, une connexion directe est établie entre les pairs pour télécharger le fichier. Plusieurs travaux sont proposés dans le contexte du partage des fichiers. La section III est consacrée à la présentation des travaux les plus importants dans cet axe.

III. Les types de système pair à pair :

Les systèmes pair à pair possèdent des caractéristiques et des objectifs communs. Mais, ces systèmes diffèrent selon les techniques et les protocoles utilisés pour la recherche et la localisation des pairs. Chaque technique possède certains avantages et inconvénients et chacune d'elles est considérée comme optimale dans un contexte, Dans cette section, nous allons

présenter les catégories des systèmes pair à pair, à savoir : Le pair à pair non structuré, le pair à pair structuré.

III.1. Les systèmes pair à pair non structurés :

Les systèmes pair à pair non structurés admettent que tous les pairs sont égaux et il n'y a pas de pairs spéciaux pour organiser le système. Dans ce système, l'emplacement des fichiers est indépendant de la structure du réseau et il n'y a pas une structure explicite à maintenir. Cette catégorie est caractérisée par une grande flexibilité, un pair peut rejoindre ou quitter le système sans influencer sa structure globale. Cependant, le manque de structure explicite rend le processus de la recherche et de la localisation plus complexe. L'organisation globale du système est comme suit:

Deux pairs qui maintiennent une connexion sont appelés « Voisins ». Chaque pair peut avoir plusieurs voisins en même temps. Le nombre de ces voisins est appelé outdegrees. Dans un système pair à pair non structuré, les messages échangés peuvent êtres des requêtes ou des résultats. Ces messages sont transmis en utilisant une connexion directe entre les deux pairs. Dans le cas contraire, s'il n'y a pas une connexion directe, le message parcourt un chemin et passe le long des autres pairs qui jouent le rôle d'un pont. La longueur du chemin parcouru par un message est appelée hop. Dans ce système, les pairs utilisent la technique de l'inondation «Flooding» pour l'échange des messages. Au niveau d'un pair, si l'utilisateur envoie une requête, ce pair va la transmettre à tous ses voisins. Chacun de ces derniers va l'envoyer à ses voisins et ainsi de suite.

III.1.1 Quelques systèmes pair à pair non structurés :

- **Gnutella** [Gnutella, 2000] est considéré comme la référence des systèmes pair à pair non structurés, spécialement dans les cas de partage de fichiers. Dans Gnutella, un pair est appelé **Servent**, parce qu'il joue le rôle d'un client et d'un serveur (**SER**Veur/cli**ENT**). La technique de Flooding est accompagnée par un TTL (Time To Live) qui détermine la longueur maximale du chemin que le message peut parcourir. La valeur par défaut du TTL est 7. A chaque fois qu'un voisin envoie un message, on décroît la valeur du TTL. L'envoie de message s'arrête lorsque la valeur de TTL est nulle. Dans la figure 16, le pair1 envoie un message à ces deux voisins (pair2, pair3). La valeur de TTL décroît à chaque passage par un pair.

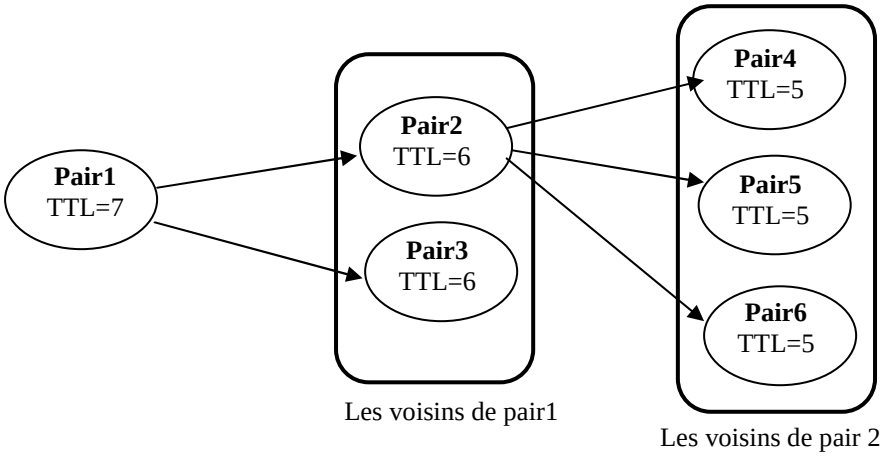


Figure 16 : Envoi de message dans un système pair à pair non structuré.

Gnutella procure une grande flexibilité dans la gestion des pairs et des ressources. Mais, le fait que tous les pairs ont des responsabilités égales, ne permet pas d'avoir des critères pour les différencier selon leurs performances ou leurs qualités. En plus, l'utilisation de la technique de flooding avec tous les voisins constitue une charge importante pour les ressources réseaux, sans garantir une meilleure qualité des résultats.

- **Super-Peer** [Yang et Garcia-Molina, 2003] Un système basé sur l'approche Super-Peer est considéré comme un système pair à pair non structuré avec la caractéristique suivante: Le réseau est divisé en plusieurs groupes, pour chacun d'eux, on désigne un pair comme serveur, et les autres comme des clients. Dans ce contexte, Les Super-Peer agissent entre eux de la même façon que dans un système pair à pair non structuré. Cependant, les autres pairs du groupe sont seulement connectés à leurs Super-Peer, sans la possibilité de faire d'autres connexions, à la manière du modèle Client/serveur. Dans la terminologie de l'approche Super-Peer, un groupe qui contient le Super-Peer et ses clients est appelé un **Cluster**. Le nombre de pairs dans chaque Cluster varie selon les paramètres de configuration et la disponibilité des participants. La figure 17 donne la topologie de ce réseau.

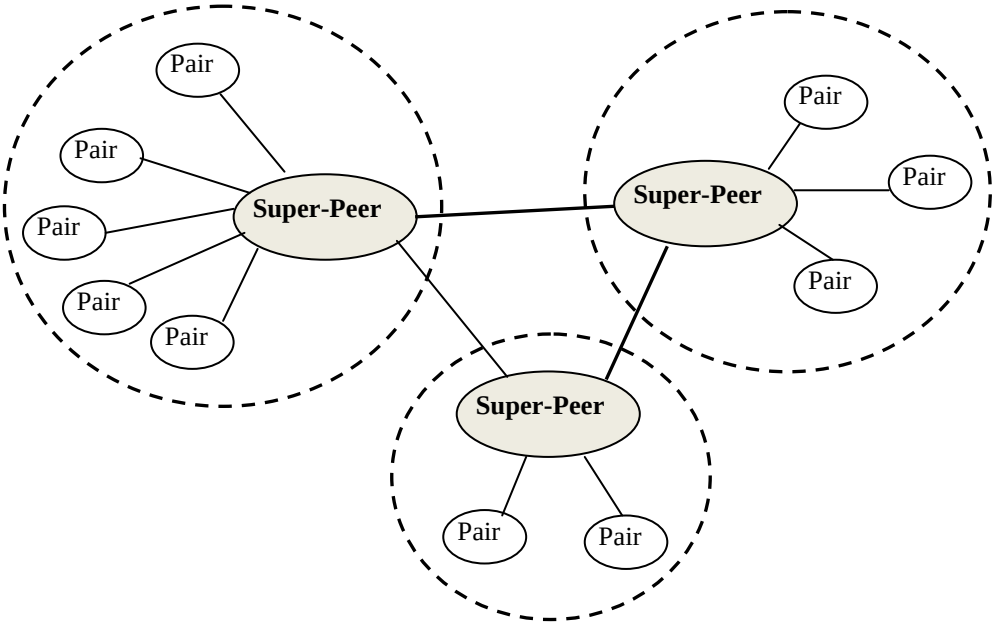


Figure 17 : Topologie du réseau Super-Peer

Dans ce système, quand le Super-Peer reçoit un message d'un autre Super-Peer voisin, il traite cette requête à son niveau sans la transmettre à ses clients. Le traitement se base sur un index local qui contient les informations nécessaires sur les ressources fournies par ses clients. Afin de maintenir cet index, chaque client qui rejoint le Cluster doit envoyer la description des ces données à son Super-Peer. En plus, les clients doivent envoyer leurs mises à jour si le client quitte le réseau, son Super-Peer supprime sa description des ressources.

Cette approche permet d'avoir :

• Tous les avantages d'un système pair à pair non structuré, spécialement entre les Super-Pairs.

• Une réduction du nombre de messages échangés.

• Une meilleure qualité des résultats en utilisant l'index local qui est géré par le Super-Peer.

Malgré ces avantages, l'aspect client/serveur qui existe dans chaque groupe apporte une faiblesse majeure. Si le Super-Peer n'est pas opérationnel pour n'importe quelles raisons (échec de fonctionnement, non disponibilité dans le système…), tous ses clients deviennent indisponibles jusqu'à l'attribution d'un nouveau Super-Peer.

Dans la catégorie des systèmes pair à pair non structurés, la meilleure approche est de trouver un compromis entre la grande flexibilité offerte par les systèmes purement non structurés comme Gnutella, et les performances fournis par les systèmes hybrides comme le Super-Peer.

III.2. Systèmes pair à pair structurés :

Dans cette section, nous nous intéressons à la catégorie des systèmes pair à pair structurés. Nous allons commencer par une brève description. Par la suite, nous allons énumérer les travaux importants qui appartiennent à cette catégorie.

III.2.1. Caractéristiques des systèmes pair à pair structurés

Dans les systèmes pair à pair structurés, l'emplacement des pairs et des fichiers partagés est associé à la structure du réseau. Les techniques de localisation et de recherches se basent sur des tables de hachage distribuées DHT (Distributed Hash Table), L'utilisation de ces DHT permet de garantir un accès optimal aux pairs et aux ressources.

L'utilisation des DHT améliore de façon significative le processus de recherche dans un système pair à pair . Cependant, la gestion de cette table qui est de nature distribuée est une tâche ardue. Spécialement si on veut garantir la caractéristique d'auto organisation, importante dans les systèmes pair à pair. En plus, la technique des tables DHT souffre de deux problèmes majeurs :

a) Les DHT nécessitent une correspondance exacte entre la requête et la ressource.

b) Un pair peut joindre ou quitter plusieurs fois le système. Cet aspect dynamique de tous les pairs nécessite une gestion lourde pour maintenir la table DHT.

III.2.2. Quelques systèmes pair à pair structurés :

Plusieurs systèmes adoptent la technique des DHT pour accomplir le processus de recherche des pairs et des ressources.

- Dans **CAN** (**C**ontenant **A**ddressable **N**etwork) [Ratnasamy et al., 2001], le réseau est considéré comme un espace cartésien multidimensionnel. L'espace est décomposé en partitions ou régions. Chaque pair est responsable d'une zone qui peut être elle-même repartitionnée ou fusionnée avec les autres zones. Chacune de ces dernières représente une clé qui correspond à la ressource partagée.

  L'idée dans CAN est basée sur la gestion dynamique de ces zones en suivant la disponibilité des pairs. Quand un pair veut rejoindre le système, il doit trouver une zone adéquate pour participer à la mise à jour de la table distribuée. Pour cela il commence par envoyer une requête auprès d'un pair connue (bootstrap). Ce dernier trouve un point dans une zone adéquate qui est déjà utilisé par un autre pair. Cette zone va être partitionnée en deux, ce qui permet aux deux pairs d'avoir leurs propres zones.

  La figure 18 présente le déroulement du processus de partitionnement dans l'espace d'adressage. L'espace est de taille 8 x 8, n_1 rejoint le système en premier, puis n_2, n_3, n_4 et n_5.

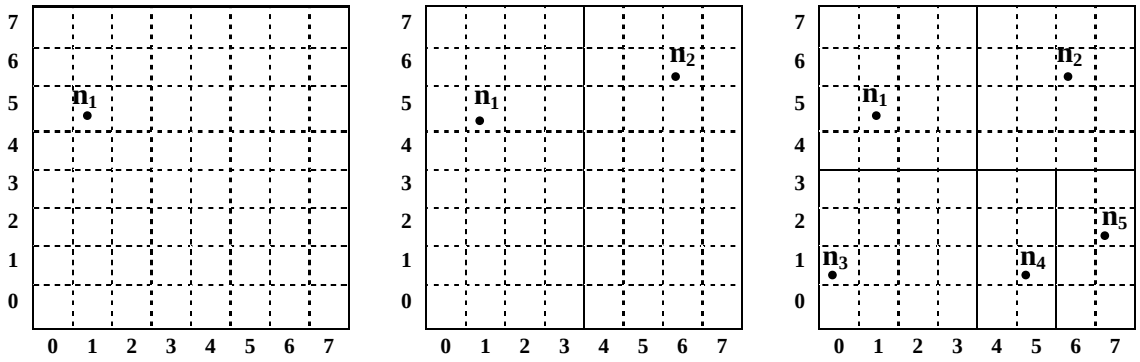


Figure 18 : Partitionnement de l'espace adressable dans CAN

Après le partitionnement de l'espace, les pairs traitent les requêtes envoyées. A chaque fois qu'un pair reçoit une requête qui ne peut pas satisfaire, il utilise sa table de routage pour la renvoyer vers un de ses voisins dont la requête correspond le plus.

- **Chord** [Storia et al., 2001] est un autre système pair à pair structuré. Chord fournit un service distribué de localisation qui permet l'insertion, la localisation et la suppression des valeurs associées à des clés spécifiques. Dans Chord, les clés représentent les pairs et les ressources, tandis que leurs valeurs correspondent à des adresses réseaux des pairs ou des fichiers. Les clés et les valeurs sont considérées comme des tableaux d'octets, dépourvues de tout sens. La sémantique est associée à l'application qui les utilise.

  Chord propose une API simple qui fournit les fonctions de bases à savoir :

  Insert, Lookup, update, join et leave. En plus, Chord utilise la redondance pour améliorer la robustesse du système. La valeur de la redondance r est paramétrée selon le degré voulu.

IV. Quelques travaux d'alignement des ontologies dans un système pair à pair :

Plusieurs travaux sont proposés dans le contexte de l'alignement des ontologies dans un environnement pair à pair. Dans ce qui suit, sont présentés trois travaux de référence, à savoir PIAZZA, EDUTELLE et SECCO.

IV.1 Le système PIAZZA :

PIAZZA est un système qui intègre les sources de données hétérogènes en se basant sur une architecture pair à pair. Les sources de données peuvent être de nature semi-structurée comme XML, ou des ontologies en RDF. Dans PIAZZA, on suppose que, malgré l'existence des langages de représentation sémantique, comme RDF ou OWL, la plupart des sources existantes sont basées sur un modèle semi structuré comme dans XML. En plus, les sources de données peuvent être des nœuds autonomes comme dans le cas du pair à pair. Pour cela,

PIAZZA est considéré comme un PDMS (Peer Data Management System), soit un système de gestion de données de pair, [Ives et al., 2004].

Dans une application PIAZZA, chaque pair peut avoir deux rôles :

a) Fournir le schéma des ces données locales.

b) Fournir le schéma ou l'ontologie qui peut l'invoquer.

La sémantique dans PIAZZA est fournie par un mapping local entre un ensemble restreint ou un groupe de pairs. Quand un schéma d'un pair est invoqué par une requête, le système peut utiliser les données des autres pairs qui possèdent un mapping avec le premier. Cela est défini comme étant un mapping chaîné.

Les caractéristiques principales du système PIAZZA sont ;

• La proposition d'un langage pour la médiation qui permet le mapping entre la structure des données et celles du domaine, ce langage est basé sur XQuery.

• La proposition d'un algorithme qui permet le traitement des requêtes. La complexité de cet algorithme réside dans le fait que le mapping est bi-directionnel (une source est décrite en fonction d'une autre et vice versa). Ainsi, l'algorithme doit proposer un mapping XML vers RDF et l'inverse, RDF vers XML en gardant la spécificité de chaque schéma.

Dans PIAZZA, le paradigme pair à pair est respecté, dans le sens où il y a une égalité des rôles entre les noeuds. En plus, la requête est propagée tout au long les pairs du même groupe, en exécutant à chaque invocation le mapping adéquat.

IV.2 Le système Edutella :

Le projet Edutella a pour objectif de fournir une infrastructure méta-données pour les applications pair à pair en se basant sur le langage RDF [Nejdl et al., 2002]. Edutella suppose que les ressources partagées par un pair peuvent être de nature complexe, et non seulement des fichiers. En fait, si les fichiers sont principalement localisés par leurs noms, ce n'est pas le cas pour les ressources éducatives (cours, articles livres…). Ces dernières nécessitent une description plus complexe.

En plus, Edutella tend d'uniformiser les services et les mécanismes déjà existants dans le domaine pair à pair. Cela, afin d'assurer un développement homogène des applications et d'avoir une interopérabilité entre les applications pair à pair hétérogènes.

Pour cela, Edutella propose un ensemble de services. Chacun d'eux s'occupe d'un aspect particulier dans l'architecture Edutella. Ces services sont principalement :

le service de requête, le service de réplication, le service de correspondance «alignement».

• Service de requête : Ce service est le plus important dans l'architecture Edutella. Il se base sur les métas-données de chaque pair pour répondre aux requêtes envoyées.

• Service de réplication : Ce service a pour but d'assurer la disponibilité et la persistance des données locales, en les répliquant dans d'autres pairs.

• Service de correspondance. Ce service assure la correspondance entre les métas-données hétérogènes, Il permet de fournir la translation des requêtes et des schémas pour assurer une interopérabilité entre les applications pair à pair.

Afin d'utiliser les métas données décrites en RDF, Edutella propose un ensemble de langages appropriés pour l'échange des requêtes, à savoir le langage RDF-QEL-i (RDF Query Exchange Language), le i désigne le numéro de la version utilisée.

Chaque nouvelle version apporte certaines extensions à la précédente. Par exemple, RDF-QEL-2 étend RDF-QUEL-1 par le support de la disjonction dans les triples.

En plus du langage supporté, Edutella est ouvert sur les autres langages proposés dans le cadre du Web sémantique. Spécialement les langages de requêtes pour RDF, comme RQL, TRIPLE et même SQL. Pour cela, Edutella propose des Wrapper qui assurent la transformation de requêtes entre ces langages et le langage RDF-QEL propre à Edutella.

Un autre aspect important dans Edutella est la topologie utilisée. Dans [Nejdl et al., 2003], Edutella suit une topologie Supper-Peer, qui procure d'avantage de performance et de qualité pour la gestion des métas données RDF.

IV.3 L'algorithme SECCO:

L'algorithme SECCO (SEmantiC Coordinator) est un système d'alignement des ontologies hétérogènes qui se base sur le mapping ad-hoc dans une architecture pair à pair [Pirrò et al., 2008], SECCO calcule le mapping entre les concepts contenus dans les ontologies des pairs. Il adopte une méthode de mapping basée sur trois mesures de similarité évaluées au moyen de trois comparateurs (matchers) différents: syntaxique, lexical et contextuel.

- Le comparateur syntaxique : il se base sur la comparaison des termes ou des chaînes de caractères ou bien les textes. Elles sont employées pour calculer la valeur de la similarité des entités textuelles, telles que des noms, des étiquettes, des commentaires présents dans les ontologies.
- Le comparateur lexical : il utilise de ressources externes pour calculer la valeur de similarité entre deux termes, tel que WordNet, qui est employé pour trouver des relations telles que la synonymie entre des termes, ou pour calculer la distance sémantique entre les termes, en utilisant des liens sémantiques dans WordNet, afin de décider s'il existe une relation entre eux.

  La similarité est décidée grâce aux liens sémantiques déjà existants dans WordNet tels que des liens synonymes (pour l'équivalence), des liens hyponymes/ hypernymes

(pour la subsomption), La métrique de Jiang et Conrath [Jiang et Conrath, 1997] produit la similarité sémantique entre deux concepts c_1 et c_2 telle que :

$$sim(c_1, c_2) = 1 - \frac{IC(c_1) + IC(c_2) - 2 * IC\big(sub(c_1, c_2)\big)}{2}$$

où les valeurs de (IC) Information Content sont obtenus par la structure de WordNet et sub (c_1, c_2) indique que le concept qui subsume c_1 et c_2 .

- Le comparateur contextuel : le contexte d'un concept est constitué par ses propriétés et de l'ensemble des concepts avec lesquels il est directement lié (les concepts voisins), cet matcher est utilisé pour raffiner les valeurs de similarité obtenu par les matchers lexical et/ou syntaxique, Il adopte la notion de contexte, les correspondances entre les concepts sont obtenus si on évaluer comment un concept c_1 de la première ontologies s'inscrit dans le contexte des concepts de la deuxième ontologie.

V. Conclusion :

L'architecture pair à pair constitue une notion importante dans notre travail. Dans ce chapitre, nous avons présenté un aperçu global du paradigme pair à pair.

Nous avons donné quelques définitions pour le terme pair à pair, Ensuite, Nous avons présenté les caractéristiques importantes, ces dernières sont principalement : la décentralisation, l'autonomie, la nature ad hoc, l'auto-organisation, la tolérance aux pannes, l'extensibilité et la montée en charge, ainsi que les objectifs de ce paradigme, a savoir, le calcul distribué, l'agrégation des ressources et le partage des fichiers. Comme les systèmes pair à pair sont divisés en plusieurs types, nous avons présenté les deux catégories des systèmes pair à pair les plus répandues. La première est la catégorie des systèmes pair à pair non structurés, la deuxième est la catégorie des systèmes pair à pair structurés. Pour chaque catégorie, nous avons donné une brève description ainsi que les principaux systèmes proposés.

Dans la quatrième section, nous nous sommes intéressés à la relation qui existe entre l'alignement des ontologies et le paradigme pair à pair. Nous avons présenté cette relation en se basant sur des exemples des systèmes pair à pair fonctionnels qui intègrent les deux notions, Nous avons présentés trois travaux de référence, à savoir PIAZZA, EDUTELLE et SECCO, Chaque système propose sa propre approche, dans le but d'améliorer sa qualité et ses performances.

Chapitre 4 :
Proposition d'un algorithme d'alignement dans un environnement pair à pair

Introduction :

Le paradigme pair à pair fournit une infrastructure importante pour le partage des ressources (fichiers, et autres). Chaque pair peut être considéré comme une source de données distribuées et hétérogènes, l'hétérogénéité est due aux différentes représentations et modèles qui sont utilisés pour la description des données du pair. Ainsi, Différentes approches ont été proposées pour mettre en œuvre l'interopérabilité de ces structures de connaissances hétérogènes. Parmi ces approches, **l'alignement d'ontologies** qui consiste à mettre en correspondance les concepts d'une ontologie, dite ontologie source (O_S), avec les concepts d'une autre ontologie, dite ontologie cible (O_c).

Nous avons évoqué dans les chapitres précédents la problématique de notre travail, à savoir, la proposition d'une solution pour l'alignement des ontologies dans les systèmes pair à pair.

L'algorithme présenté ci-dessous permet de mettre en œuvre une stratégie de partage de correspondances entre les pairs, et de leur permettre de collaborer pour découvrir des correspondances en utilisant la connaissance dont il dispose chaque pair. Pour cela, Étant donné un pair source et un pair cible, cet algorithme calcule l'alignement des concepts parmi les concepts contenus dans les ontologies. Il adopte une stratégie d'alignement basée sur l'exploitation des mesures de similarité floue simples et des mesures de similarité floue combinées.

Notre objectif est de proposer une solution qui se divise en deux parties :

a) la coordination entre les pairs dans un environnement pair à pair.

b) et l'algorithme d'alignement (flou matcher) qui s'exécute au niveau de chaque pair dans cet environnement.

Ce chapitre est organisé comme suit : nous commençons par un aperçu global sur l'approche proposée. Par la suite, nous présentons quelques définitions que nous allons les utiliser comme outils dans l'algorithme proposé pour l'alignement des ontologies dans un environnement pair à pair, Par la suite, nous présentons la stratégie d'alignement qui se divise en deux parties, Le chapitre se termine par une conclusion.

I. Motivation et Objectif :

Les architectures pair à pair se présentent actuellement comme une solution viable pour permettre le partage de ressources à l'échelle de l'Internet. En effet, aussi bien d'un point de vue commercial que scientifique, les architectures pair à pair suscitent un véritable engouement. Le paradigme du pair à pair garantit un fonctionnement à large échelle [Oram, 2001]. Un très grand nombre de pairs peut interagir dans le réseau, de manière à permettre le partage d'une grande quantité de ressources. Aussi appelé d'égal à égal, chaque participant à un système pair à pair peut être à la fois client et serveur. Le fonctionnement du système ne repose sur aucune coordination centralisée.

Ainsi, le comportement global du réseau résulte uniquement des interactions locales entre les pairs qui se connectent et se déconnectent.

Le paradigme pair à pair est devenu un des plus importants domaines de l'informatique actuelle, particulièrement dans l'axe de l'informatique distribuée et Internet. Cela est dû principalement à la nature du pair à pair qui comporte plusieurs avantages, comme l'auto-organisation, la tolérance aux pannes, l'autonomie, l'aspect dynamique …

En plus de ces caractéristiques, le pair à pair propose des solutions dans plusieurs domaines d'applications tels que le calcul distribué, le partage de fichiers, l'agrégation des ressources…, pour toutes ces raisons, le concept du pair à pair a suscité beaucoup d'intérêt dans la communauté scientifique et professionnelle.

Notre objectif est de proposer une solution qui se divise en deux parties :

a) La coordination entre les pairs dans un environnement pair à pair.

b) la proposition d'un algorithme d'alignement d'ontologies (flou matcher) qui s'exécute au niveau de chaque pair dans notre environnement, cet algorithme se base sur l'exploitation des mesures de similarité floue simples et des mesures de similarité floue combinées.

II. Architecture de l'alignement dans un environnement pair à pair:

II.1 la coordination entre les pairs :

II.1.1 Quelques définitions :

Définition 1 *(pair source et pair cible):* un pair source est un pair sémantique qui envoie une requête sur le réseau pair à pair à des pairs cibles qui reçoivent la demande, l'exécutent localement et retournent comme résultat l'ensemble des correspondances entre les différents concepts.

Définition 2 *(concept):*Un concept ontologique c peut représenter un objet, une idée. Un concept peut être divisé en trois parties : le terme, la notion et l'ensemble d'objets.

Définition 3 *(requête)*: Soit O une ontologie, une requête est de la forme RQ = < s > où s ∈ C est un concept qui appartient à une ontologie de pair source.

Définition 4 (*Les méthodes de comparaison ou matcher)*: Un matcher est une fonction utilisée pour calculer la distance entre deux entités. Les matchers sont des fonctions qui peuvent être combinées dans le processus de matching.

Définition 5 (*ensemble des correspondances entre les concepts) :* Soit la requête de pair source RQ = <s>, et l'ontologie O_c du pair cible, un concept de l'alignement A entre chaque concept de pair cible $p \in C_c$ et le concept de pair source $s \in C_s$ est un ensemble de 3-tuples de la forme <s, p, μ > où $\mu \in [0,1]$ est la valeur de similarité floue entre les concepts s et p.

Définition 6 (*seuil de similarité th):* Th est utilisé pour filtrer les résultats d'alignement des concepts, l'utilisation de filtre à base de seuil de similarité est pour réduire le nombre de fausses correspondances entre les concepts des différentes ontologies à aligner.

II.1.2 L'algorithme de coordination entre les pairs:

Cet algorithme conçus pour faire des alignements entre les ontologies dans un réseau pair à pair, Étant donné un pair source et un pair cible, cet algorithme calcule l'alignement des concepts parmi les concepts contenus dans les ontologies. Il adopte une stratégie d'alignement basée sur l'exploitation des mesures de similarité floue simple et combinée.

L'objectif de l'algorithme présenté ci-dessous est de mettre en œuvre une stratégie de partage de correspondances entre les pairs, et de leur permettre de collaborer pour découvrir des correspondances en utilisant la connaissance dont dispose ces pairs.

Chaque pair possède une ontologie exploitée pour conceptualiser et modéliser un ensemble des connaissances dans un domaine donné. et joue un double rôle:

(1) *pair source*: quand il envoie une requête au réseau;

(2) *pair cible*: quand il reçoit la demande (la requête du pair source) il exécute localement l'algorithme proposé.

Les pairs cibles utilisent l'algorithme pour obtenir les correspondances entre un concept source s et de leurs concepts de l'ontologie locale. A chaque fois qu'un pair cible reçoit une requête, il exécute l'algorithme avec une entrée de la forme suivante: $I = <s, O_c, Th>$

Où: **s** est le concept de pair source;

O_c est l'ontologie de pair cible;

Th ∈ [0,1] est une valeur qui exprime le seuil de similarité entre deux concepts, Th est utilisé pour filtrer les résultats d'alignement des concepts et pour écarter les résultats d'alignement avec une valeur de similarité très basse.

Le pseudo-code de l'algorithme de coordination entre les pairs dans un environnement pair à pair est représenté dans la Figure suivante:

Input: An input $I=<c_s, O_c, Th>$ where $O_c = (C,R)$

Output: les alignements des concepts A

Method:

```
1. A=∅ ;
2. for each c∈C do
3.    simf=evaluer_floue_similarité(cs,c); //voir la section I.3.3
4.    sim=simf; // valeur de similarité combinée
5.    if sim>Th then
6.       a.s=cs
7.       a.ci=c;
8.       a.V=sim;
9.       A=A∪ a;
10.   end-if
11. end-for
12. return A;
```

Figure19: Le pseudo-code de l'algorithme de coordination entre les pairs

La figure 20 représente l'approche globale de l'algorithme proposé. Un pair source envoi une requête RQ = <s> a tous les pairs du réseau, cette requête contient un concept c_s de l'ontologie source O_s. Cette demande atteint les pairs cibles qui exécutent l'algorithme à son niveau. Le module d'agrégation (ou de combinaison) de l'algorithme exploite la fonction de similarité par la combinaison des valeurs de similarité fourni par les comparateurs et filtre les résultats en fonction du Th (le seuil de similarité). Pius, l'algorithme retourne les valeurs de similarité des correspondances entre le concept source et les concepts cibles au pair source, qui stocke ces résultats dans l'espace de stockage des alignements.

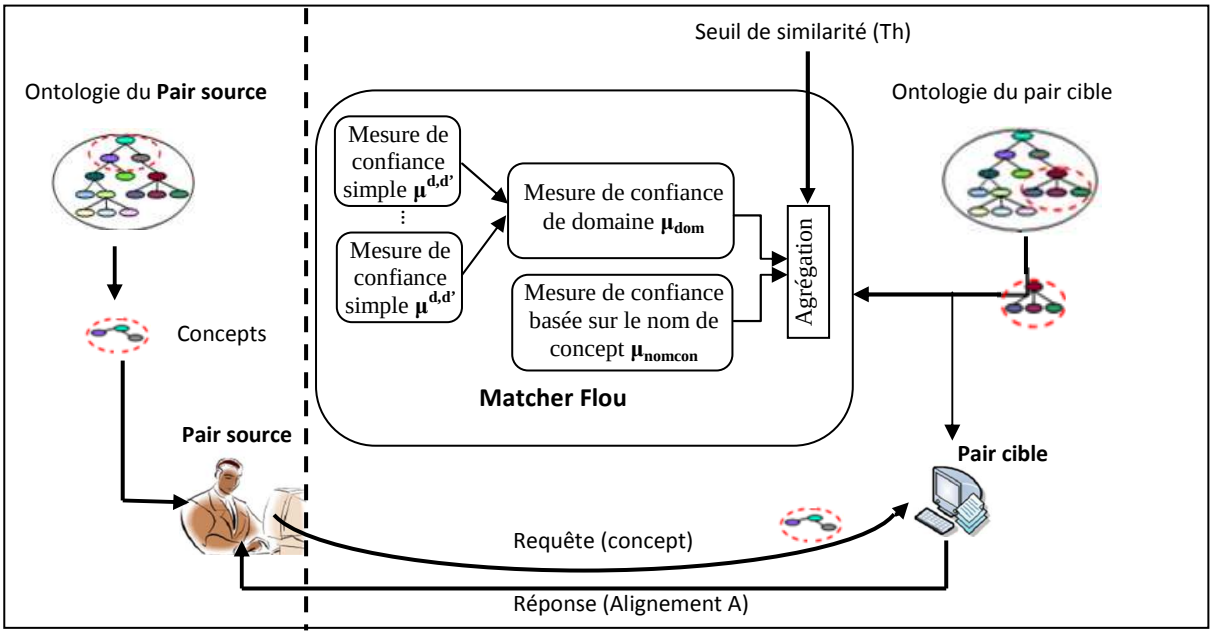


Figure 20 : l'architecture de l'algorithme d'alignement dans un environnement pair à pair

II.2 L'alignement des ontologies :

L'alignement d'ontologies est un processus de découverte des correspondances entre deux ontologies (une ontologie source et une ontologie cible). Il est, généralement, décrit comme une application de l'opérateur MATCH [Rahm et Bernstein, 2001]**,** dont l'entrée est constituée d'un ensemble d'ontologies et la sortie, formée des correspondances entre ces ontologies.

On trouve dans la littérature plusieurs algorithmes qui implémentent cet opérateur. Dans cette partie de notre travail, nous allons présentés un programme pour le calcul de la similarité entre les concepts basée sur les relations floues, Notre approche est inspirée par les travaux de Fagin et Wimmers [Fagin et Wimmers, 1997] et Fagin [Fagin, 1999], qui a proposé une méthode de combinaison des réponses aux requêtes sur des sources de données différents en utilisant des concepts de la théorie des ensembles flous, pour permettre aux utilisateurs de fixer des poids

dans leurs requêtes (pour présenter leurs préférences), et les travaux de Gal et ses collèges [Gal et al., 2005], qui ont proposés un modèle pour l'intégration des bases de donnée hétérogènes.

Partie 1 :

Dans cette partie nous allons présenter deux familles d'opérateurs flous:

Les opérateurs de norme triangulaires et les opérateurs d'agrégation, qui sont généralement utilisés pour combiner les différents degrés d'appartenance flous :

II.2.1 Les opérateurs de norme triangulaire :

Une norme triangulaire (t-norme) T : [0,1] x [0,1] → [0,1] est un opérateur binaire (étend la conjonction ordinaire) ayant les propriétés suivantes, pour chaque $x, y, z \in [0,1]$:

- L'existence de l'élément neutre : T(x, 1) = T(1, x) = x,
- Monotonie (non décroissance par rapport à chaque argument):

  si $x \leq y$ alors $T(x, z) \leq T(y, z)$,
- Commutativité : T(x, y)= T(y,x),
- Associativité : T(x, T(y, z)) = T(T(x, y),z)

Il existe une infinité de normes et les plus courantes sont donnés dans la table suivante :

Norme : T(x,y)	Nom
min(x,y)	Zadeh (T_m)
x.y	Probabiliste (T_p)
max(x+y-1,0)	Lukasiewicz (T_l)

Table 7: Principales Normes Triangulaires

Il est à noter que T_m est la seule norme **idempotente** (c-à-d : $T_m(x,x)=x$)

Les normes triangulaires sont utilisées comme des opérateurs de conjonction (**et**) dans les calculs incertains.

II.2.2 Les opérateurs d'agrégation flou:

L'opérateur d'agrégation floue est une fonction f , $f : [0,1]^n \rightarrow [0,1]$ satisfait les axiomes suivantes pour chaque $x_1,\ldots,x_n \in [0,1]$:

$f(x_1,x_1,\ldots,x_1) = x_1$ (idempotence)

pour chaque $y_1,y_2,\ldots,y_n \in [0,1]$ avec $x_i \leq y_i$:

$f(x_1,x_2,\ldots,x_n) \leq f(y_1,y_2,\ldots,y_n)$ (Monotonie croissante)

f est une fonction continue (continuité)

soit $\bar{x}$ =(x_1,...,x_n) un vecteur tel que $x_i \in [0,1]$ pour tout : $1 \leq i \leq n$

et soit $\bar{\bar{w}} = (\overline{w_1},..., \overline{w_n})$ le vecteur des poids tel que : $\sum_{i=1}^{n} \overline{w_i} = 1$

Quelques exemples d'opérateurs d'agrégation floue on trouve:

La moyenne arithmétique tel que :

$$f_a(\bar{x}) = \frac{1}{n} \sum_1^n \bar{x_i} \quad \text{.....(1)}$$

Et la moyenne arithmétique pondérée tel que :

$$f_{ap}(\bar{x}, \bar{\bar{w}}) = \bar{x}.\bar{\bar{w}} \quad \text{.....(2)}$$

La moyenne arithmétique est un cas particulier de la moyenne arithmétique pondérée :

$$(\overline{w_1} = \ldots = \overline{w_n} = \frac{1}{n})$$

Il est à noter que T_m (T-norme) est également un opérateur d'agrégation flou en raison de son idempotence, ainsi que, le T-norme et l'opérateur d'agrégation flou sont comparables en utilisant l'inégalité suivante:

$$Min(x_1,\ldots,x_n) \leq f(x_1,\ldots,x_n)$$

pour tous les $x_1,...,x_n \in [0, 1]$.

Partie 2 :

II.2.3. Le programme :

Le modèle proposé, qui sera donnée en détail par la suite, utilise un programme flou pour quantifier l'imprécision et modéliser l'incertitude dans le processus d'alignement entre les ontologies. Étant donné deux ensembles de concepts A et A', nous associons une mesure de confiance entre ces deux concepts, normalisée entre 0 et 1, avec un alignement entre les concepts de A et A'.

Par conséquent, étant donné deux concepts $a \in$ A et $a' \in$ A', on dit que nous sommes μ-confident dans le mapping de a et a' (notée $a \sim \mu_{con} a'$) pour préciser notre croyance dans la qualité du mapping. Nous supposons qu'un matching manuel est un processus parfait, résultant une correspondance nette, avec $\mu_{con} = 1$.

Donc, nous présentons un programme pour le calcul de la similarité entre les concepts des ontologies basée sur les relations floues, un ensemble flou sur un domaine D (appelé aussi univers ou référenciel) est un ensemble caractérisé par la fonction d'appartenance δ_A: D→ [0, 1], ou $\delta_A(a) = \mu$ représente le degré d'appartenance flou de l'élément a dans A.

Donc $\delta_A(a)$ exprime dans quelle mesure (à quelle point) l'élément a appartient à l'ensemble flou A, quand $\delta_A(a)$ est nul, a n'appartient pas du tout à l'ensemble flou A, et quand il vaut 1, a est complètement dans A (dans ce cas a est aussi dit l'élément typique de A).

Plus $\delta_A(a)$ est proche de 1 (Resp 0) plus (Resp moins) a appartient à A, un ensemble usuel peut être vu comme un cas particulier d'ensemble flou dont la fonction d'appartenance ne prend que les valeurs 0 et 1.

Soient les domaines suivants : $D_1,D_2,\ldots,D_n$ et leurs produit cartésien : $D=D_1 \times D_2 \times \ldots \times D_n$, une relation floue R sur les domaines $D_1,D_2,\ldots,D_n$ est un ensemble flou d'éléments (tuplets) de D, $\mu^{d1,d2,\ldots,dn}$ représente le degré d'appartenance des tuplets $(d_1,d_2,\ldots,d_n)$ dans $\mathcal{R}$.

Nous allons ensuite introduire des relations de confiance, que nous les utilisons pour calculer les Mesures de confiance simples (section I.2.3.1) et les mesures de confiance combinées (section I.2.3.2).

II.2.3.1 Les *Mesures de confiance simples*:

Soient les domaines D et D', une relations de confiance simple (Primitive confidence relation) est une relation floue sur DxD', notée $\sim\mu$, ou μ (ou $\mu^{d,d'}$) est le degré d'appartenance de (d,d') dans $\sim\mu$ (noté la mesure de confiance de mapping des concepts d et d'), la mesure de confiance de la relation de confiance simple est calculée en utilisant une certaine métrique entre les concepts .

Certaines propriétés souhaitables d'une relation de confiance simple sont les suivantes :

Réflexivité: $\mu^{d,d} = 1$. Réflexivité assure que le mapping exacte reçoit le score le plus élevé possible (comme dans le cas de deux concepts identiques, par exemple, les concepts qui ont le même nom).

Symétrie : $\mu^{d,d'} = \mu^{d',d}$: La symétrie garantit que l'ordre dans lequel sont comparés deux schémas n'a aucun influence sur le résultat final.

Transitivité : $\mu^{d,d''} \geq \max_{d' \in D'} \min [\mu^{d,d'}, \mu^{d',d''}]$: ce type de transitivité est connu comme la propriété max-min transitivité [Klir et Yuan, 1995], elle fournit une base pour la génération d'une relation d'équivalence, par exemple, on peut générer α-coupe d'équivalence, qui contient tous les pairs ou la mesure de confiance est plus grande que α.

La transitivité est difficile à réaliser.

a) La Mesure de confiance basée sur les instances numériques:

Soient les domaines (l'ensemble des instances d'un concept donné) $D=\{d_1,d_2,\ldots,d_n\}$ et $D'=\{d'_1,d'_2,\ldots,d'_n\}$, on suppose que la mesure de confiance est calculée en fonction de leur distance euclidienne, normalisée entre 0 et 1 tel que:

$$\mu^{d,d'} = 1 - \frac{|d-d'|}{\max d_i \in D, d'_j \in D' \{|d_i - d'_j|\}} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots(3)$$

Cette relation de confiance simple, avec sa mesure de confiance $\mu^{d,d'}$ associe (voir l'équation (3)) est :

Réflexive (d-d = 0), et symétrique (|d – d'| = |d'− d|), et non-transitive, ce qui résulte une relation de proximité.

Soient les deux domaines numériques D et D' tel que :D={8,10,12,14,16,18} et D'={9,11,13,16,18}, la mesure de confiance de 14 (dans D) et de 16 (dans D') on appliquant la formule (3) on obtient comme résultat:

$$\mu^{d,d'} = 0.8$$

b) La Mesure de confiance basée sur les instances non numériques:

Soient les domaines (l'ensemble des instances non numériques d'un concept donné) $D=\{a_1,a_2,\ldots,a_n\}$ et $D'=\{a'_1,a'_2,\ldots,a'_n\}$, on suppose que la mesure de confiance est fondée sur les chaines de caractères (de chaque instance), normalisée entre 0 et 1 tel que:

$$\mu^{a,a'} = \frac{|a \cap a\prime|}{\max\{|a|,|a\prime|\}} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(4)$$

Cette relation de confiance simple, avec sa mesure de confiance $\mu^{a,a'}$ associe (voir l'équation (4)) est :

Réflexive, puisque pour deux instances qui ont le même nom, la longueur du sous chaine commune est la même, et donc : $\mu^{a,a'} = 1$

et symétrique (car $|a \cap a'| = |a' \cap a|$ et: $\max\{|a|,|a'|\} = \max\{|a'|,|a|\}$),

Mais, elle est non-transitive, ce qui résulte une relation de proximité.

c) La Mesure de confiance basée sur le nom du concept:

Soient O et O' deux ontologies avec ses concepts source et cible, et soient $\mathcal{A}$ et $\mathcal{A}'$ deux chaines de caractères qui représentent deux concepts de ces ontologies, et soit $\sim \mu_{nomcon}$ une

relation de confiance simple sur $O \times O'$ ou : $\sim \mu_{nomcon}$ s'appelle Mesure de confiance basée sur les noms des concepts, le calcul de cette mesure est généralement fondée sur les chaines de caractères (substring matching), tel que :

$$\mu_{nomcon}^{A,A'} = \frac{|A \cap A'|}{\max\{|A|,|A'|\}} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(5)$$

$|A \cap A'|$ Correspond à la longueur de la plus longue sous chaine commune entre A et A'.

Cette relation de confiance simple, avec sa mesure de confiance $\mu_{nomcon}^{A,A\prime}$ associe (voir l'équation 5) est :

Réflexive, puisque pour deux concepts qui ont le même nom (par exemple le concept « department » de l'ontologie university.OWL et de l'ontologie universityBench.OWL) la longueur du sous chaine commune est le nom du concept, et donc : $\mu_{nomcon}^{A,A} = 1$,

et symétrique (car $|A \cap A'| = |A' \cap A|$ et: $\max\{|A|, |A'|\} = \max\{|A'|, |A|\}$),

Mais, elle est non-transitive, ce qui résulte une relation de proximité.

Une autre méthode pour calculer la Mesure de confiance basée sur le nom du concept est de diviser la longueur de la plus longue chaîne commune par la longueur du premier (ou, du second) nom du concept, est comme suit :

$$\mu_{nomcon}^{A,A'} = \frac{|A \cap A'|}{|A|} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(6)$$

Comme un exemple on prend deux concepts de l'ontologie university.OWL et de l'ontologie universityBench.OWL :

« SystemStaff » et « staff » la mesure de confiance de ces deux concepts quand on applique la formule (6) on obtient comme résultat:

$$\mu_{nomcon}^{A,A'} = 0.4545$$

II.2.3.2 Les Mesures de confiance combinées :

Les mesures de confiance combinées utilisent des mesures de confiance (soit simple ou combinées) pour calculer une nouvelle mesure de confiance (voir la figure 21). Dans cette partie nous allons présentés deux mesures de confiance combinées :

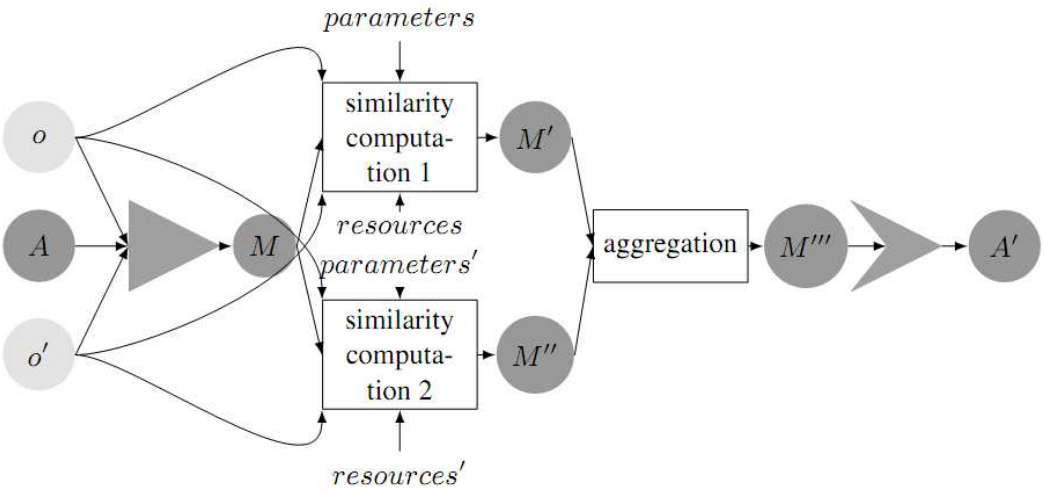


Figure 21: Mesure de confiance composée par agrégation

[Euzenat et Shvaiko, 2007]

a) La Mesure de confiance de domaine :

Est une méthode pour calculer la valeur de la mesure de confiance pour les domaines numériques et non négatifs (l'ensemble des instances), nous pouvons calculer la mesure de confiance pour faire un mapping de deux domaines on se basant sur le mapping de leurs instances (avec le calcul de la mesure de confiance simple basée sur les valeurs).

Soient D et D' deux domaines. Et soit μ_{dom} une fonction de mapping basée sur la mesure de confiance de domaine. $\sim\mu_{dom}$ est un mapping de domaine basé sur les mesures de confiance de domaine. μ_{dom} est le mapping calculé a partir de mesures de confiance simple de chaque paire d'instances de D et D' .

On peut calculer μ_{dom} comme suit :

$$\mu_{dom}^{D,D'} = \min_{d\in D, d'\in D'}(\mu^{D,d'}, \mu^{D',d}) \quad \text{.....(7)}$$

Où, pour tout : $d' \in D'$,

$$\mu^{D,d'} = \max_{d\in D}(\mu^{d,d'}) \quad \text{.....(8)}$$

et pour tout : $d \in D$,

$$\mu^{D',d} = \max_{d'\in D'}(\mu^{d,d'}) \text{.....(9)}$$

Où: $$\mu^{d,d'} = 1 - \frac{|d-d'|}{\max d_i \in D, d'_j \in D'\{|d_i - d_j|\}} \text{.....(10)}$$

Chaque valeur dans D est aligner avec «la meilleure valeur » dans D', et vice versa, et la force de μ_{dom} est déterminée par la force de «la relation la plus faible »

Notre utilisation de min et max est en ligne avec les conventions de la logique floue, Où le max est interprété comme la disjonction et min est interprété comme la conjonction.

Exemple : Soient les deux domaines numériques D et D' tel que : D={8,10,12} et D'={9,11,12}, la mesure de confiance de ces deux domaines, on appliquant la formule (7) on obtient:

$$\mu_{dom}^{D,D'} = 0.75$$

Cette relation de confiance des domaines est une relation de proximité car elle est:

Réflexive : D = D' et pour chaque élément : $d' \in D$ on a :

$$\mu^{D,d'} = \max_{d \in D}(\mu^{d,d'})$$

$$= \mu^{d,d}$$

$$=1$$

Et donc : $$\mu_{dom}^{D,D} = \min_{d \in D, d' \in D}(\mu^{D,d'}, \mu^{D,d}) = 1$$

Et **symétrique** :

$$\mu_{dom}^{D,D'} = \min_{d \in D, d' \in D'}(\mu^{D,d'}, \mu^{D',d})$$

$$= \min_{d\prime \in D\prime, d \in D}(\mu^{D\prime,d}, \mu^{D,d\prime})$$

$$= \mu_{dom}^{D',D}$$

Alors: $$\mu_{dom}^{D,D'} = \mu_{dom}^{D',D}$$

Ce qui résulte une relation de proximité.

De la même manière on peut utiliser cette méthode pour calculer la mesure de confiance pour les domaines non numériques (avec un ensemble des instances non numériques) :

$$\mu_{dom}^{A,A'} = \min_{a\in A, a'\in A'}(\mu^{A,a'}, \mu^{A',a}) \quad \text{.....(11)}$$

Où, pour tout : $a' \in A'$,

$$\mu^{A,a'} = \max_{a\in A}(\mu^{a,a'}) \quad \text{.....(12)}$$

et pour tout : $a \in A$,

$$\mu^{A',a} = \max_{a'\in A'}(\mu^{a,a'}) \quad \text{.....(13)}$$

Où: $$\mu^{a,a'} = \frac{|a \cap a'|}{\max\{|a|,|a'|\}}$$

b) La Mesure de confiance combinée d'un concept:

La mesure de confiance pour le mapping des concepts est déterminée par la combinaison ou l'agrégation des mesures de confiance basée sur le nom d'un concept (μ_{nomcon}) et des mesures de confiance de domaine (μ_{dom}).

Par conséquent, étant donné deux concepts A et A', avec leurs domaines D et D', respectivement, la mesure de confiance de concept de A et A', Notée μ_{con}, est une fonction qui se présente comme suit :

$$\mu_{con}^{A,A'} = h1(\mu_{nomcon}^{A,A'}, \mu_{dom}^{D,D'}) \quad \text{.....(14)}$$

$h1$ est appelé l'opérateur d'agrégation flou.

$h1$ est la moyenne arithmétique (average operator) des deux mesures de similarité μ_{nomcon} et μ_{dom}.

III. Conclusion :

Nous avons décrit dans ce chapitre l'algorithme proposé pour l'alignement des ontologies dans un environnement pair à pair, cet algorithme permet de mettre en œuvre une stratégie de partage de correspondances entre les pairs, et de leur permettre de collaborer pour découvrir des correspondances en utilisant les ontologies dont il dispose chaque pair (pair source et pair cible), cet algorithme calcule l'alignement des concepts parmi les concepts contenus dans les ontologies. Avec une stratégie d'alignement basée sur l'exploitation (i) des mesures de similarité floue simple: basée sur les instances numériques $\mu^{d,d'}$ et non numériques $\mu^{a,a'}$, et des Mesures basées sur le nom de concept μ_{nomcon} (ii) et des mesures de similarité floue combinée : qui combinent les mesures de confiance de l'ensemble des instances (domaine μ_{dom}) et de concept μ_{con}.

Notre solution se divise en deux parties :

i) la coordination entre les pairs dans un environnement pair à pair.

ii) l'algorithme d'alignement qui s'exécute au niveau de chaque pair dans cet environnement.

Chapitre 5 :
Implémentation et Evaluation

Afin de montrer l'intérêt de l'architecture et de la solution décrite dans le chapitre précédent, nous avons conçu notre programme avec NetBeans IDE[1] version 6.9.1, qui permet de montrer la technique utilisée pour l'alignement des ontologies.

Notre programme est réalisé en 6 étapes décrites comme suit :

Etape 1 : chargement des ontologies : cette technique est basée sur l'API jena[2] qui exploite les données du modèles OWL et RDF dans notre application Java, une fois notre ontologie est lit, il faut la valider avec un moteurs d'inférences, le chargement ce fait à travers ce code :

```
import com.hp.hpl.jena.ontology.OntClass;
import com.hp.hpl.jena.ontology.OntModel;
import com.hp.hpl.jena.ontology.OntResource;
import com.hp.hpl.jena.rdf.model.ModelFactory;
import com.hp.hpl.jena.vocabulary.OWL;

  model = ModelFactory.createOntologyModel( PelletReasonerFactory.THE_SPEC );

      // crée le  model de l'ontologie
      System.out.print( "Lecture ..." );
      model.read( ontology );
      System.out.println( "Lecture terminer" );
```

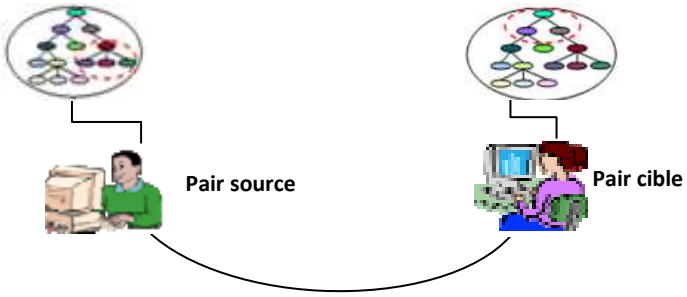


Figure 22 : processus de chargement des ontologies

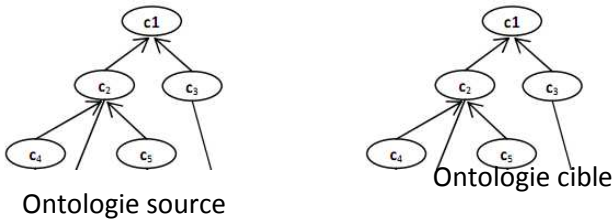
[1] http://netbeans.org

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

Etape 2 : Validation des ontologies on utilisant le moteur d'inférence Pellet[1] (qui permet de raisonner au niveau terminologique et sur les instances de concepts), avec extraction des concepts et des instances de chaque ontologie, cette étape est implémentée par le code suivant :

```
import org.mindswap.pellet.jena.PelletInfGraph;
import org.mindswap.pellet.jena.PelletReasonerFactory;
// chargement du  model pour le reasoner
      System.out.print( "Preparer..." );
      model.prepare();
      System.out.println( "terminer" );

      // classification et validation
      System.out.print( "Classification..." );
      ((PelletInfGraph) model.getGraph()).getKB().classify();
      System.out.println( "terminer" );
```

Après les étapes 1 et 2 nous obtiendrons deux arbre corresponds à nos ontologies chargées, ou les concepts et les instances sont validés.



Etape 3: Cette étape est appelée l'étape de normalisation (le niveau linguistique) : avant de calculer les valeurs de similarité entre les concepts, les chaines de caractères (des concepts) sont nettoyées, pour que les résultats de la comparaison des chaines seront améliorés.

Cette phase de normalisation est utilisée dans notre algorithme comme suit :

- Normalisation des caractères: convertit toutes les majuscules dans une chaîne de caractères en leurs formes minuscules.
- Normalisation des espaces: remplace toutes les séquences consécutives des espaces, trouvées dans une chaîne de caractères par un seul caractère d'espace.
- Suppression des chiffres.
- Élimination des ponctuations.

[1] http://www.mindswap.org/2003/pellet

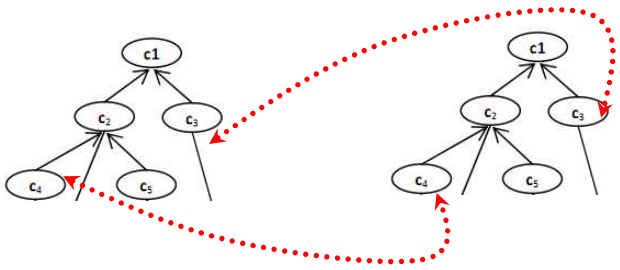
Cette étape est réalisée avec le code suivant :

```
public static String replaceW(String text, String searchString, String replacementString) {
   StringBuffer sBuffer = new StringBuffer();
   int pos = 0;
   while ((pos = text.indexOf(searchString)) != -1) {
   sBuffer.append(text.substring(0, pos) + replacementString);
   text = text.substring(pos + searchString.length());
   }
   sBuffer.append(text);
   return sBuffer.toString();
   }
      Concept =replaceW(concept,".","");
      concept =replaceW(concept ,"(","");
      concept =replaceW(concept ,")","");
      concept =replaceW(concept ,"[","");
      ……..
      concept =replaceW(concept ,"]","");
```

Etape 4: Calcul des mesures de confiances simples et combinées pour chaque concept et instance (présentées dans le chapitre précédent), dans une première matrice qui contient les concepts des deux ontologies chargées ainsi une deuxième matrice qui contient les instances.

Etape 5 : Détermination du seuil de similarité Th : Th est utilisé pour filtrer les résultats d'alignement des concepts. Le filtrage à base de seuil est la seule méthode adaptée par les travaux existants pour réduire le nombre des fausses correspondances.

Etape 6: l'algorithme retourne les valeurs de similarité du mapping entre le concept source et les concepts cibles, et les différentes correspondances qui peuvent être stocké dans l'espace de stockage de mapping.



I. Description du programme réalisé :

I.1 Menu principal:

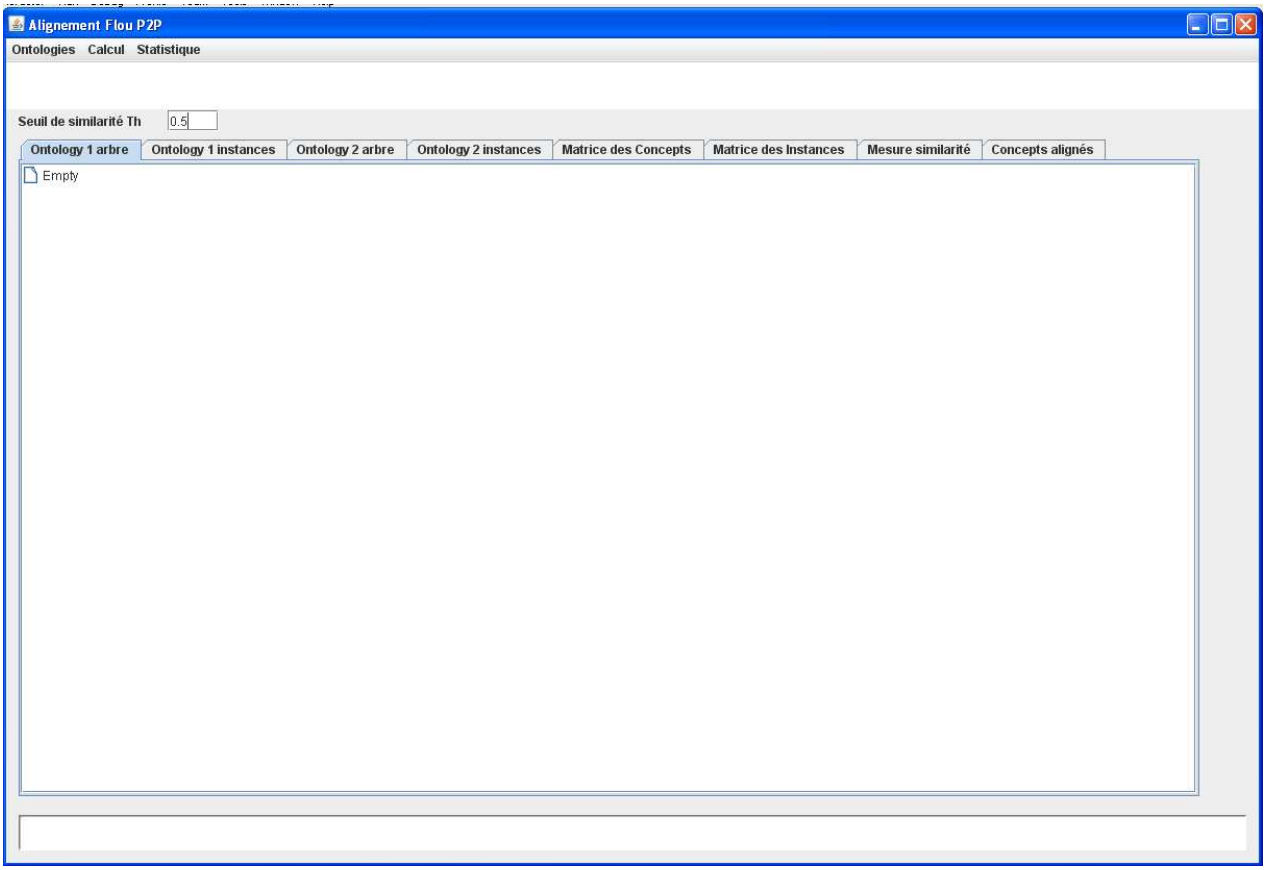


Figure 23: Le menu principal de notre programme

La barre de menu contient 3 éléments, le rôle de chaque élément est expliqué comme suit :

a) **Ontologie :** pour le chargement des ontologies (l'ontologie source **Ontologie 1** et l'ontologie cible **Ontologie 2**).

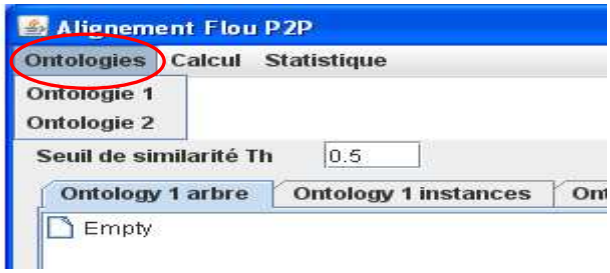


Figure 24: L'élément «Ontologies» de la barre de menu

b) Calcul : pour calculer les mesures de similarité simples ou combinées.

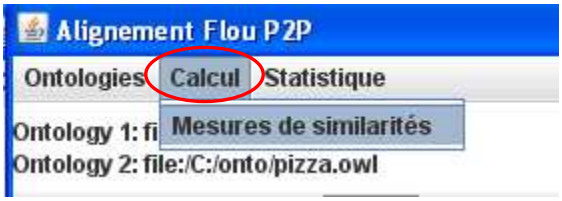


Figure 25: L'élément «Calcul» de la barre de menu

c) Statistique : pour afficher les statistiques sur les mesures calculées ainsi le nombre des correspondances trouvés, nombre de concepts, instances, seuil de similarité, les concepts alignés, nombres concepts alignés.

La fenêtre des onglets contient les éléments suivants :

1- Ontology 1 arbre: pour afficher l'arbre de la 1ère ontologie.
2- Ontology 1 instances: pour afficher l'ensemble des instances de la 1ère ontologie.
3- Ontology 2 arbre: pour afficher l'arbre de la 2ème ontologie.
4- Ontology 2 instances: pour afficher l'ensemble des instances de la 2éme ontologie.
5- Matrice des concepts: pour afficher la matrice des concepts des deux ontologies.
6- Matrice des instances: pour afficher la matrice des instances des deux ontologies.
7- Mesure de similarité: pour afficher la mesure de similarité finale qui combine les valeurs de la mesure de similarité de concept et d'instances.
8- Concepts alignés: pour afficher la matrice des concepts qui sont filtrés et alignés, avec la valeur de la mesure de similarité.

I.2 Chargement des ontologies:

On respectant les étapes de la réalisation de notre programme, la première étape est le chargement des deux ontologies par notre programme qui permet d'afficher les concepts de chaque ontologie comme suit :

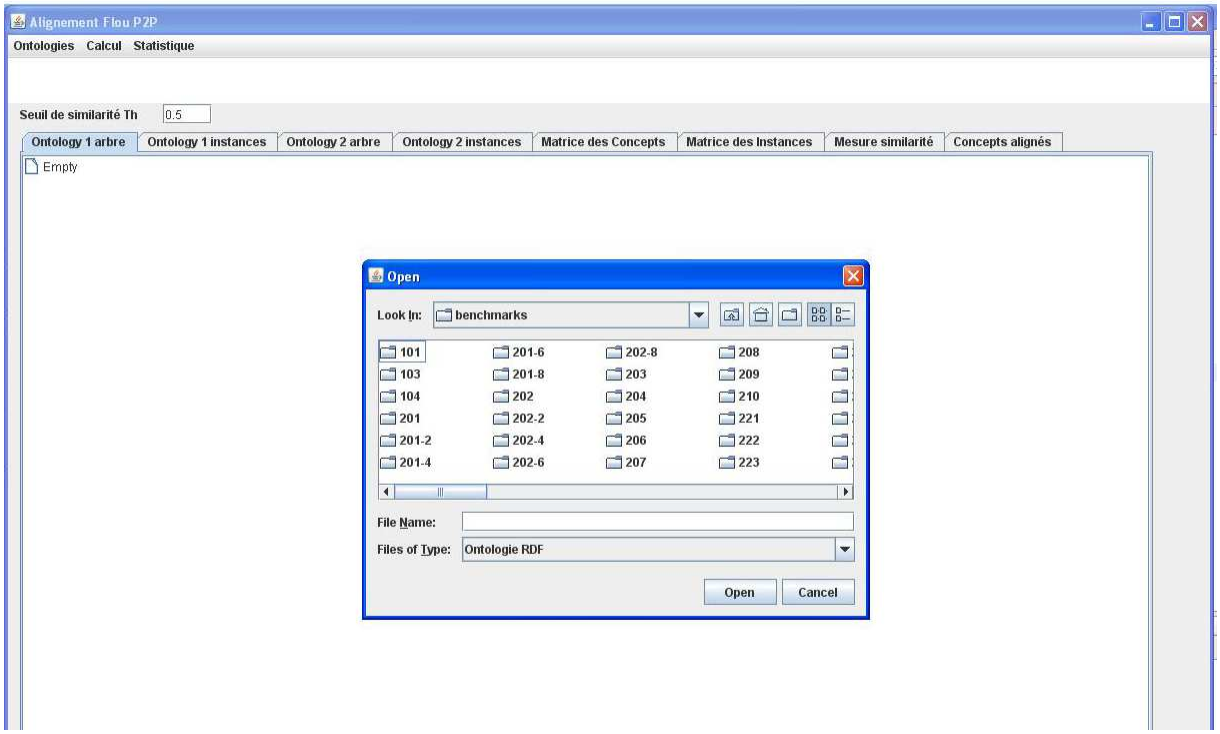


Figure 26: Chargement des ontologies sélectionnées

Exemple: l'ontologie 1 : bench70\benchmarks\101\ onto.rdf,

l'ontologie 2 : bench70\benchmarks\228\ onto.rdf

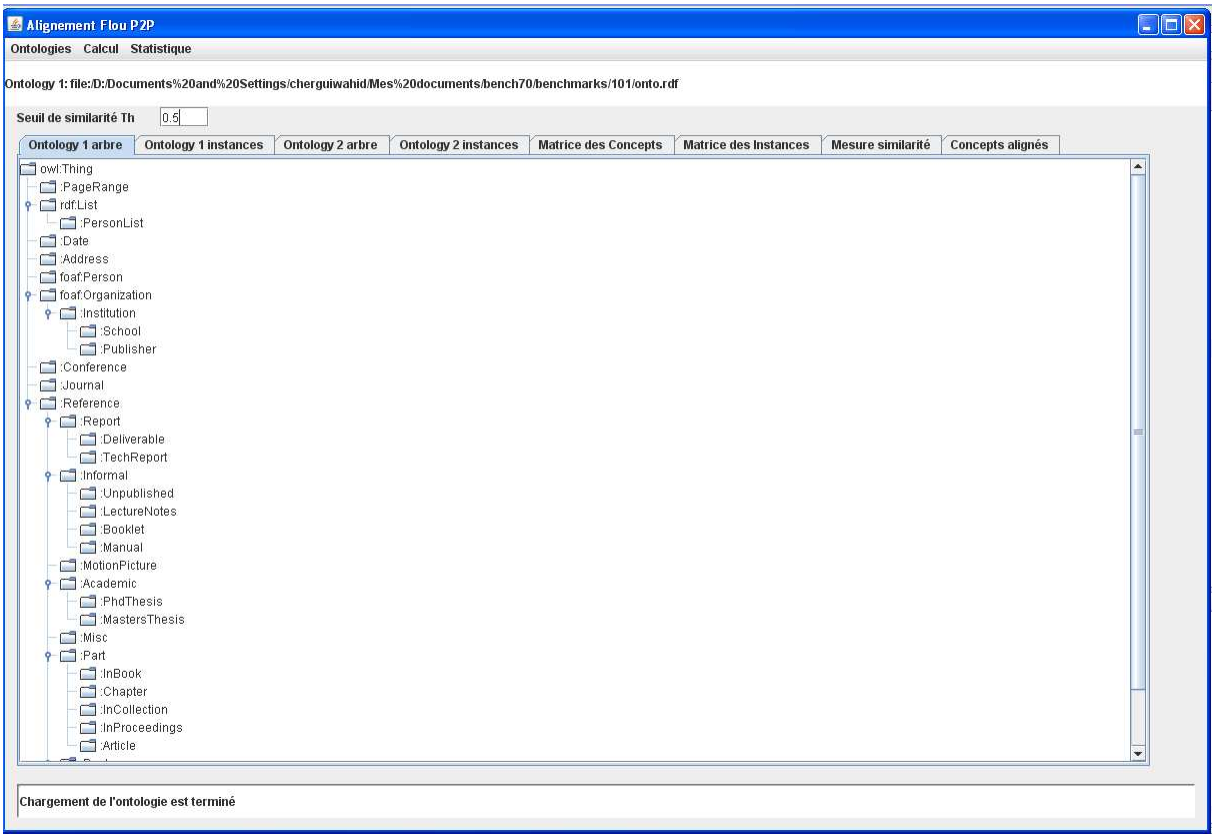


Figure 27: Chargement de l'ontologie 1 « onto.rdf »

Les instances de la 1ère ontologie :

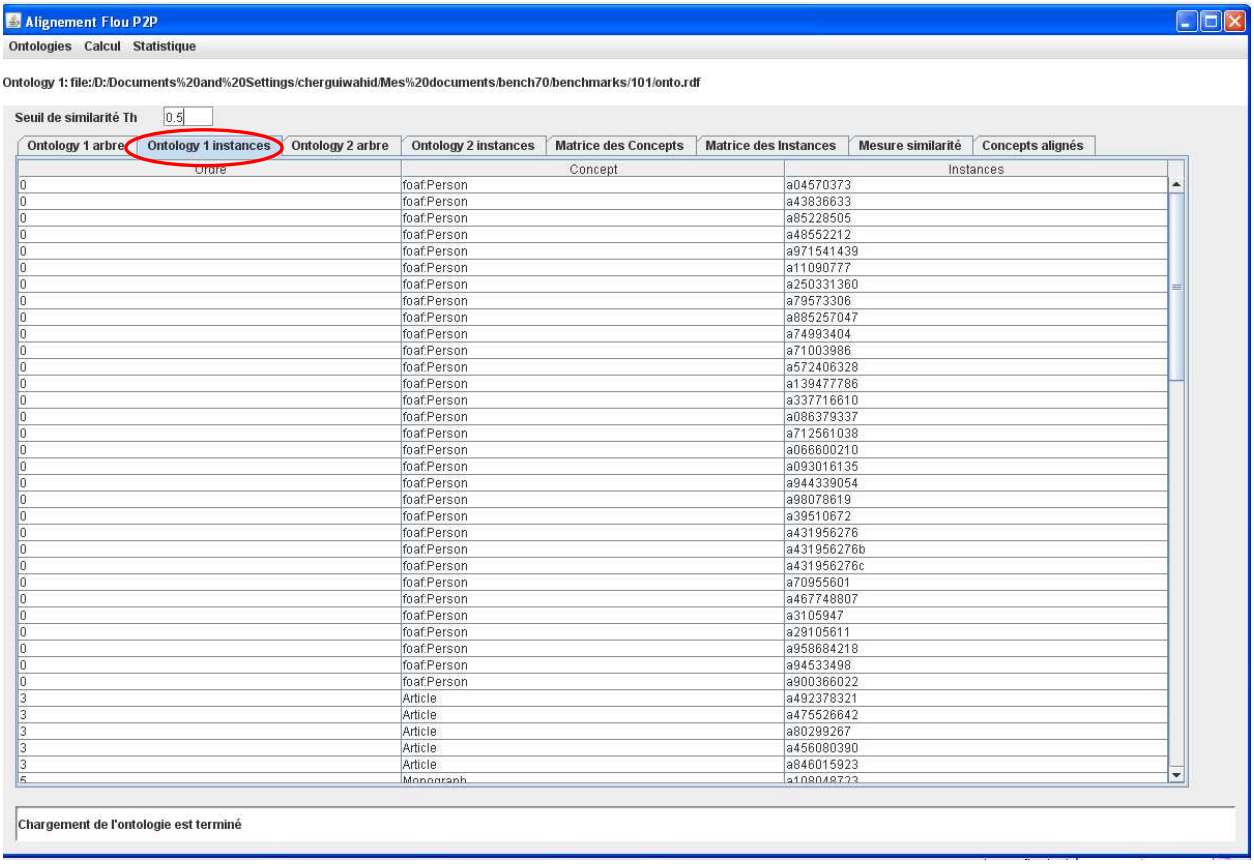


Figure 28: Les instances de l'ontologie 1 « onto.rdf »

Ordre	Concept	Instance
0	foaf:Person	a04570373
0	foaf:Person	a43836633
0	foaf:Person	a85228505
0	foaf:Person	a48552212
0	foaf:Person	a971541439
0	foaf:Person	a11090777
0	foaf:Person	a250331360
0	foaf:Person	a79573306
0	foaf:Person	a885257047
0	foaf:Person	a74993404
0	foaf:Person	a71003986
0	foaf:Person	a572406328
0	foaf:Person	a139477786
…	…	…

Table 8 : Les instances de l'ontologie 1 « onto.rdf »

Ontologie 2 :

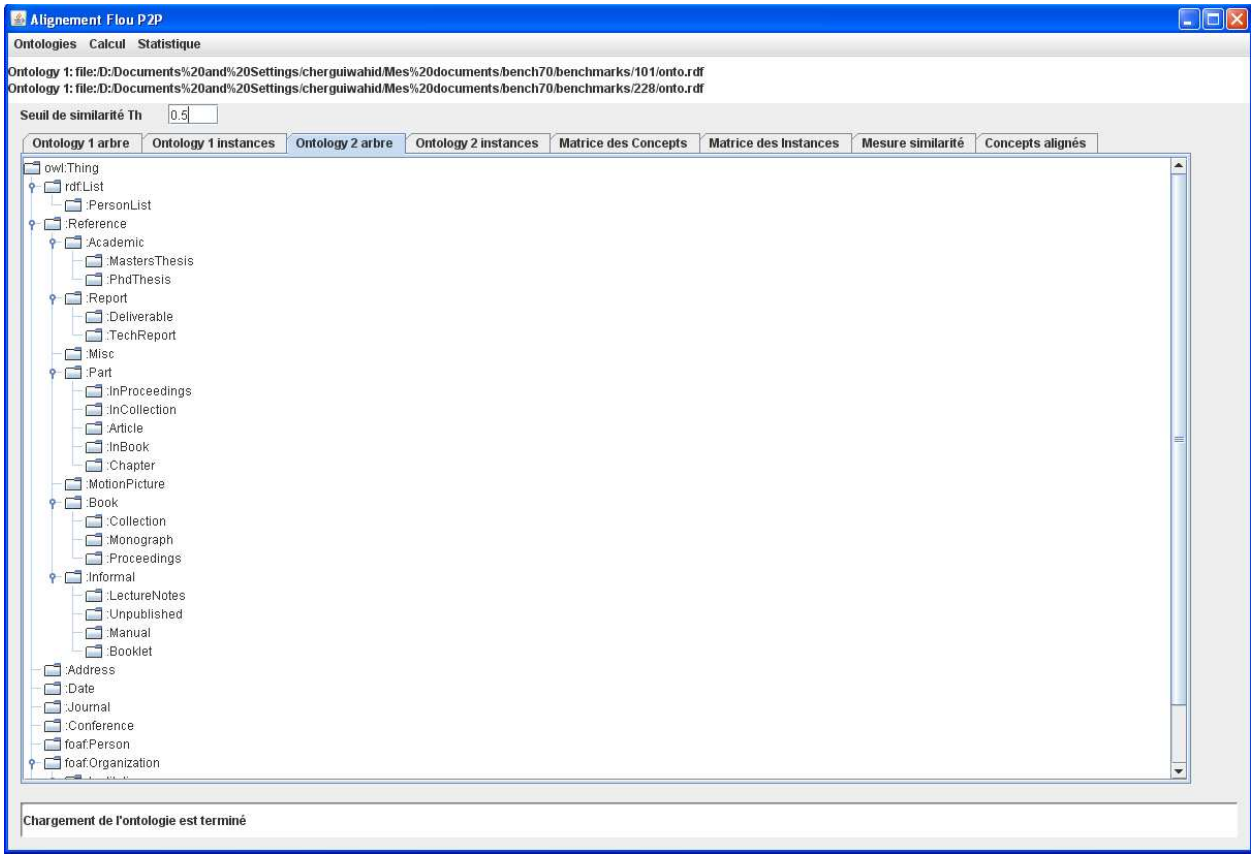


Figure 29: Chargement de l'ontologie 2 « onto.rdf »

Les instances de la 2ème ontologie :

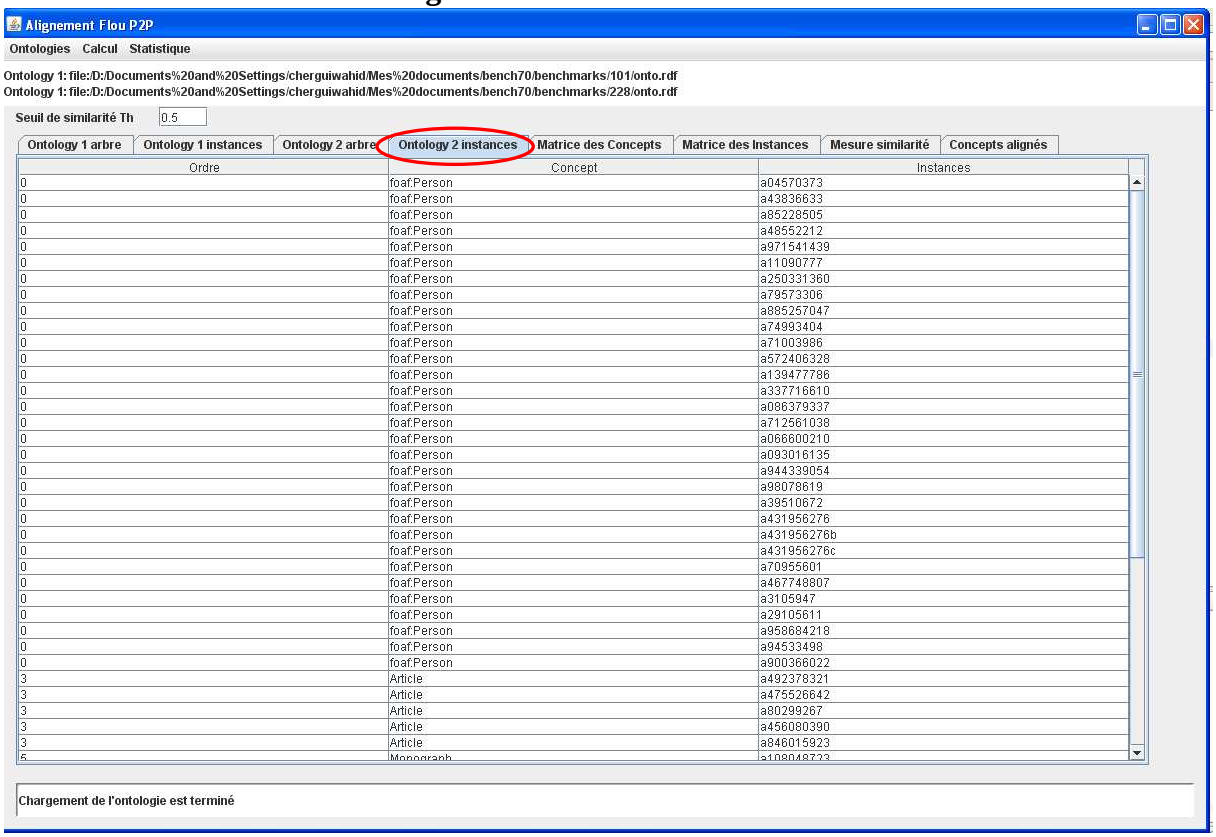


Figure 30: Les instances de l'ontologie 2 « onto.rdf »

Ordre	Concept	Instance
0	foaf:Person	a04570373
0	foaf:Person	a43836633
0	foaf:Person	a85228505
0	foaf:Person	a48552212
0	foaf:Person	a971541439
0	foaf:Person	a11090777
0	foaf:Person	a250331360
0	foaf:Person	a79573306
0	foaf:Person	a885257047
0	foaf:Person	a74993404
0	foaf:Person	a71003986
0	foaf:Person	a572406328
0	foaf:Person	a139477786
0	foaf:Person	a337716610
0	foaf:Person	a086379337
0	foaf:Person	a712561038
…		

Table 9 : Les instances de l'ontologie 2 « onto.rdf »

I.3 Calcul des mesures de similarité :

On calcule les mesures de similarité à partir de la barre de menu «calcul» → «Mesures de similarités», comme suit :

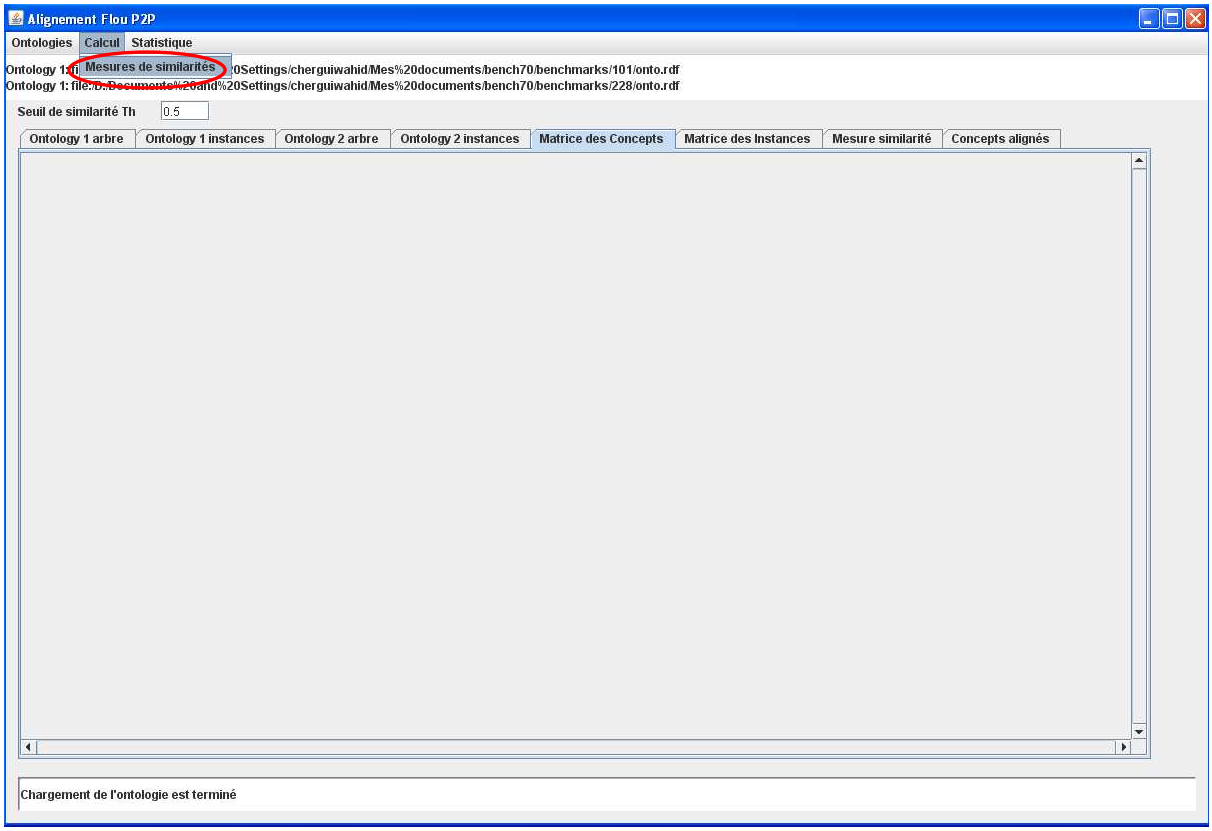
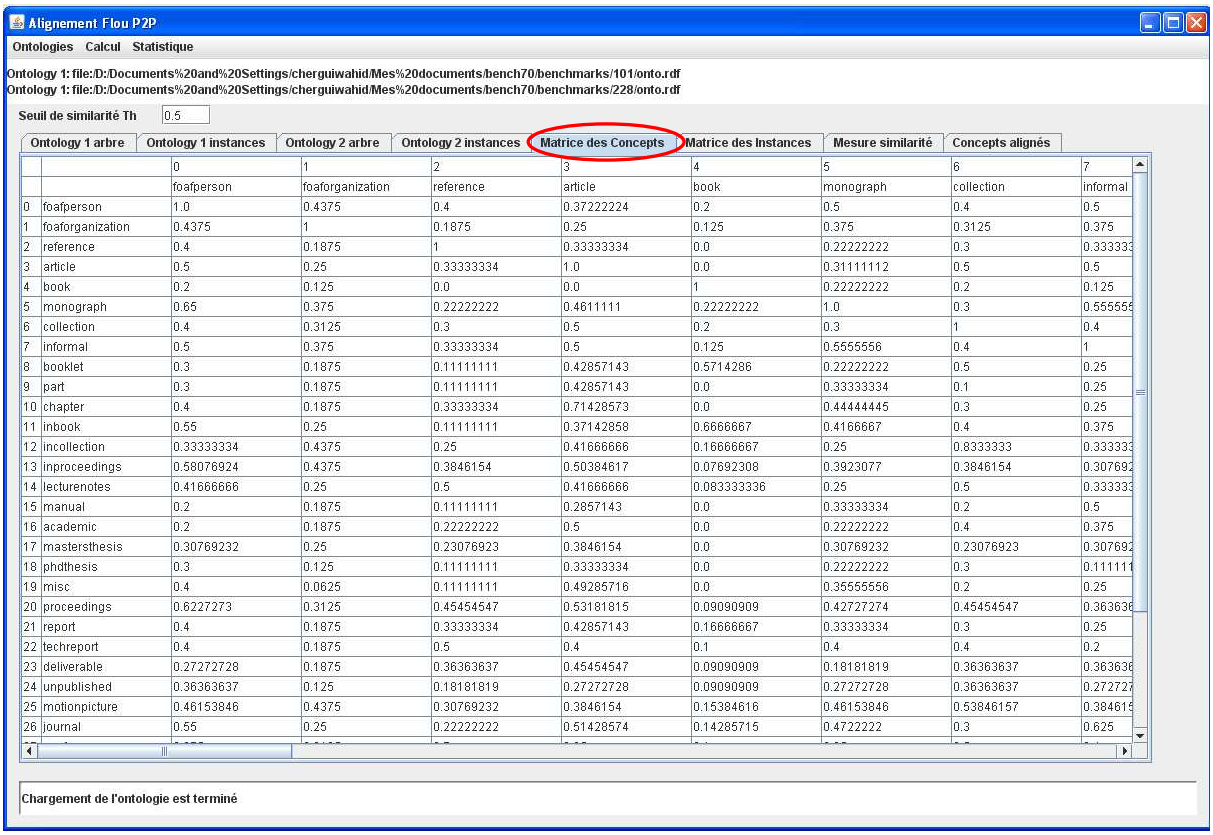


Figure 31: Calcul des mesures de similarité

I.4 Matrice des concepts : Le calcul des différentes valeurs de la matrice des concepts avec l'utilisation des mesures de similarité simples:



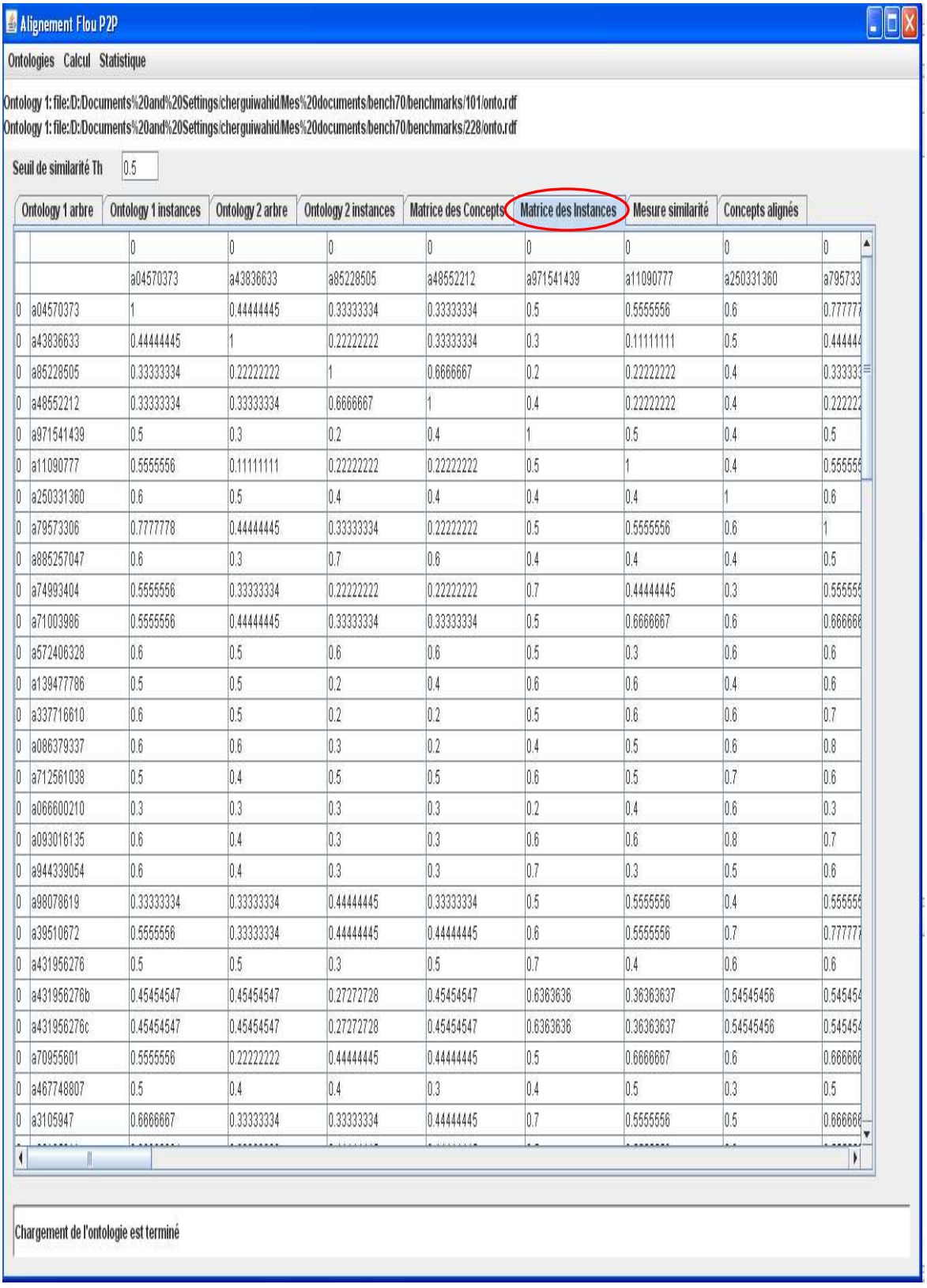
		0	1	2	3	4	5	6	7
		foafperson	foaforganization	reference	article	book	monograph	collection	informal
0	foafperson	1.0	0.4375	0.4	0.37222224	0.2	0.5	0.4	0.5
1	foaforganization	0.4375	1	0.1875	0.25	0.125	0.375	0.3125	0.375
2	reference	0.4	0.1875	1	0.33333334	0.0	0.22222222	0.3	0.33333
3	article	0.5	0.25	0.33333334	1.0	0.0	0.31111112	0.5	0.5
4	book	0.2	0.125	0.0	0.0	1	0.22222222	0.2	0.125
5	monograph	0.65	0.375	0.22222222	0.4611111	0.22222222	1.0	0.3	0.55555
6	collection	0.4	0.3125	0.3	0.5	0.2	0.3	1	0.4
7	informal	0.5	0.375	0.33333334	0.5	0.125	0.5555556	0.4	1
8	booklet	0.3	0.1875	0.11111111	0.42857143	0.5714286	0.22222222	0.5	0.25
9	part	0.3	0.1875	0.11111111	0.42857143	0.0	0.33333334	0.1	0.25
10	chapter	0.4	0.1875	0.33333334	0.71428573	0.0	0.44444445	0.3	0.25
11	inbook	0.55	0.25	0.11111111	0.37142858	0.6666667	0.4166667	0.4	0.375
12	incollection	0.33333334	0.4375	0.25	0.41666666	0.16666667	0.25	0.8333333	0.33333
13	inproceedings	0.58076924	0.4375	0.3846154	0.50384617	0.07692308	0.3923077	0.3846154	0.30769
14	lecturenotes	0.41666666	0.25	0.5	0.41666666	0.083333336	0.25	0.5	0.33333
15	manual	0.2	0.1875	0.11111111	0.2857143	0.0	0.33333334	0.2	0.5
16	academic	0.2	0.1875	0.22222222	0.5	0.0	0.22222222	0.4	0.375
17	mastersthesis	0.30769232	0.25	0.23076923	0.3846154	0.0	0.30769232	0.23076923	0.30769
18	phdthesis	0.3	0.125	0.11111111	0.33333334	0.0	0.22222222	0.3	0.11111
19	misc	0.4	0.0625	0.11111111	0.49285716	0.0	0.35555556	0.2	0.25
20	proceedings	0.6227273	0.3125	0.45454547	0.53181815	0.09090909	0.42727274	0.45454547	0.36363
21	report	0.4	0.1875	0.33333334	0.42857143	0.16666667	0.33333334	0.3	0.25
22	techreport	0.4	0.1875	0.5	0.4	0.1	0.4	0.4	0.2
23	deliverable	0.27272728	0.1875	0.36363637	0.45454547	0.09090909	0.18181819	0.36363637	0.36363
24	unpublished	0.36363637	0.125	0.18181819	0.27272728	0.09090909	0.27272728	0.36363637	0.27272
25	motionpicture	0.46153846	0.4375	0.30769232	0.3846154	0.15384616	0.46153846	0.53846157	0.38461
26	journal	0.55	0.25	0.22222222	0.51428574	0.14285715	0.4722222	0.3	0.625

Figure 32: Calcul de la matrice des concepts

	ordre	0	1	2	3	4	…
ordre	**les concepts**	foafperson	foaforganization	reference	article	book	
0	foafperson	1	0.4375	0.4	0.37222224	0.2	
1	foaforganization	0.4375	1	0.1875	0.25	0.125	
2	reference	0.4	0.1875	1	0.33333334	0	
3	article	0.5	0.25	0.33333334	1	0	
4	book	0.2	0.125	0	0	1	
5	monograph	0.65	0.375	0.22222222	0.4611111	0.22222222	
6	collection	0.4	0.3125	0.3	0.5	0.2	
7	informal	0.5	0.375	0.33333334	0.5	0.125	
8	booklet	0.3	0.1875	0.11111111	0.42857143	0.5714286	
9	part	0.3	0.1875	0.11111111	0.42857143	0	
10	chapter	0.4	0.1875	0.33333334	0.71428573	0	
11	inbook	0.55	0.25	0.11111111	0.37142858	0.6666667	
12	incollection	0.33333334	0.4375	0.25	0.41666666	0.16666667	
13	inproceedings	0.58076924	0.4375	0.3846154	0.50384617	0.07692308	
14	lecturenotes	0.41666666	0.25	0.5	0.41666666	0.08333334	
15	manual	0.2	0.1875	0.11111111	0.2857143	0	
…							

Table 10 : Calcul de la matrice des concepts

I.5 Le calcule de la matrice des instances:



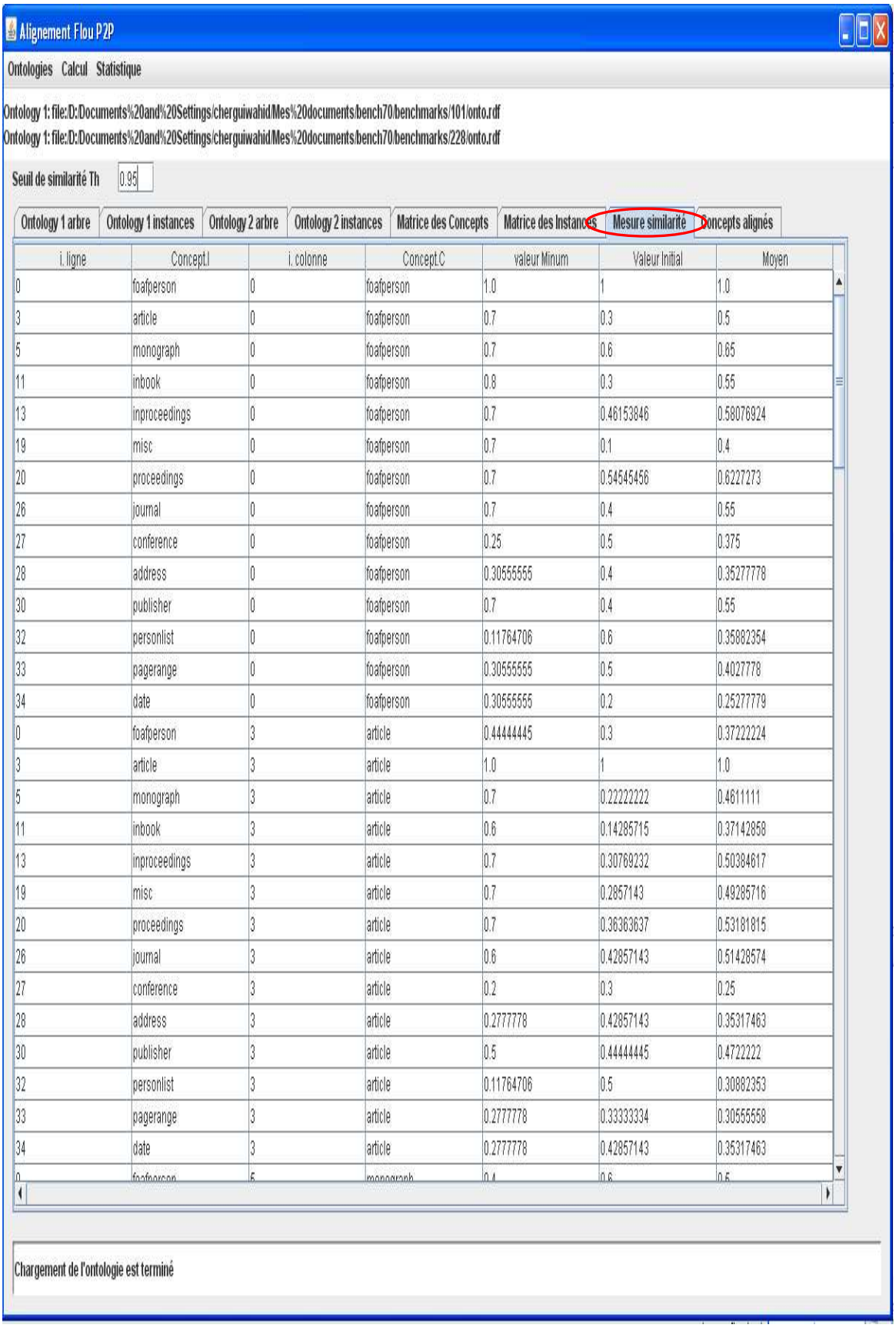
		0	0	0	0	0	0	0	0
		a04570373	a43836633	a85228505	a48552212	a971541439	a11090777	a250331360	a795733
0	a04570373	1	0.44444445	0.33333334	0.33333334	0.5	0.5555556	0.6	0.777777
0	a43836633	0.44444445	1	0.22222222	0.33333334	0.3	0.11111111	0.5	0.444444
0	a85228505	0.33333334	0.22222222	1	0.6666667	0.2	0.22222222	0.4	0.333333
0	a48552212	0.33333334	0.33333334	0.6666667	1	0.4	0.22222222	0.4	0.222222
0	a971541439	0.5	0.3	0.2	0.4	1	0.5	0.4	0.5
0	a11090777	0.5555556	0.11111111	0.22222222	0.22222222	0.5	1	0.4	0.555555
0	a250331360	0.6	0.5	0.4	0.4	0.4	0.4	1	0.6
0	a79573306	0.7777778	0.44444445	0.33333334	0.22222222	0.5	0.5555556	0.6	1
0	a885257047	0.6	0.3	0.7	0.6	0.4	0.4	0.4	0.5
0	a74993404	0.5555556	0.33333334	0.22222222	0.22222222	0.7	0.44444445	0.3	0.555555
0	a71003986	0.5555556	0.44444445	0.33333334	0.33333334	0.5	0.6666667	0.6	0.666666
0	a572406328	0.6	0.5	0.6	0.6	0.5	0.3	0.6	0.6
0	a139477786	0.5	0.5	0.2	0.4	0.6	0.6	0.4	0.6
0	a337716810	0.6	0.5	0.2	0.2	0.5	0.6	0.6	0.7
0	a086379337	0.6	0.6	0.3	0.2	0.4	0.5	0.6	0.8
0	a712561038	0.5	0.4	0.5	0.5	0.6	0.5	0.7	0.6
0	a066600210	0.3	0.3	0.3	0.3	0.2	0.4	0.6	0.3
0	a093016135	0.6	0.4	0.3	0.3	0.6	0.6	0.8	0.7
0	a944339054	0.6	0.4	0.3	0.3	0.7	0.3	0.5	0.6
0	a98078619	0.33333334	0.33333334	0.44444445	0.33333334	0.5	0.5555556	0.4	0.555555
0	a39510672	0.5555556	0.33333334	0.44444445	0.44444445	0.6	0.5555556	0.7	0.777777
0	a431956276	0.5	0.5	0.3	0.5	0.7	0.4	0.6	0.6
0	a431956276b	0.45454547	0.45454547	0.27272728	0.45454547	0.6363636	0.36363637	0.54545456	0.545454
0	a431956276c	0.45454547	0.45454547	0.27272728	0.45454547	0.6363636	0.36363637	0.54545456	0.545454
0	a70955601	0.5555556	0.22222222	0.44444445	0.44444445	0.5	0.6666667	0.6	0.666666
0	a467748807	0.5	0.4	0.4	0.3	0.4	0.5	0.3	0.5
0	a3105947	0.6666667	0.33333334	0.33333334	0.44444445	0.7	0.5555556	0.5	0.666666

Figure 33: Calcul de la matrice des instances

	Ordre	**0**	**0**	**0**	**0**	**0**	**0**	…
Ordre	**Instances**	a04570373	a43836633	a85228505	a48552212	a971541439	a11090777	…
0	a04570373	1	0.444444	0.333333	0.3333333	0.5	0.555556	
0	a43836633	0.44444445	1	0.222222	0.3333333	0.3	0.111111	
0	a85228505	0.33333334	0.222222	1	0.6666667	0.2	0.222222	
0	a48552212	0.33333334	0.333333	0.666667	1	0.4	0.222222	
0	a971541439	0.5	0.3	0.2	0.4	1	0.5	
0	a11090777	0.5555556	0.111111	0.222222	0.2222222	0.5	1	
0	a250331360	0.6	0.5	0.4	0.4	0.4	0.4	
0	a79573306	0.7777778	0.444444	0.333333	0.2222222	0.5	0.555556	
0	a885257047	0.6	0.3	0.7	0.6	0.4	0.4	
0	a74993404	0.5555556	0.333333	0.222222	0.2222222	0.7	0.444444	
…	…							

Table 11 : Calcul de la matrice des instances

I.6 Les mesures de la similarité : dans cette partie, les valeurs des mesures de similarité calculées précédemment (min-max des mesures de similarité des instances et les mesures de similarité du nom de concept) sont affichées dans cette matrice avec le calcul des mesures de similarité finales (la moyenne arithmétique) :



i. ligne	Concept.I	i. colonne	Concept.C	valeur Minum	Valeur Initial	Moyen
0	foafperson	0	foafperson	1.0	1	1.0
3	article	0	foafperson	0.7	0.3	0.5
5	monograph	0	foafperson	0.7	0.6	0.65
11	inbook	0	foafperson	0.8	0.3	0.55
13	inproceedings	0	foafperson	0.7	0.46153846	0.58076924
19	misc	0	foafperson	0.7	0.1	0.4
20	proceedings	0	foafperson	0.7	0.54545456	0.6227273
26	journal	0	foafperson	0.7	0.4	0.55
27	conference	0	foafperson	0.25	0.5	0.375
28	address	0	foafperson	0.30555555	0.4	0.35277778
30	publisher	0	foafperson	0.7	0.4	0.55
32	personlist	0	foafperson	0.11764706	0.6	0.35882354
33	pagerange	0	foafperson	0.30555555	0.5	0.4027778
34	date	0	foafperson	0.30555555	0.2	0.25277779
0	foafperson	3	article	0.44444445	0.3	0.37222224
3	article	3	article	1.0	1	1.0
5	monograph	3	article	0.7	0.22222222	0.4611111
11	inbook	3	article	0.6	0.14285715	0.37142858
13	inproceedings	3	article	0.7	0.30769232	0.50384617
19	misc	3	article	0.7	0.2857143	0.49285716
20	proceedings	3	article	0.7	0.36363637	0.53181815
26	journal	3	article	0.6	0.42857143	0.51428574
27	conference	3	article	0.2	0.3	0.25
28	address	3	article	0.2777778	0.42857143	0.35317463
30	publisher	3	article	0.5	0.44444445	0.4722222
32	personlist	3	article	0.11764706	0.5	0.30882353
33	pagerange	3	article	0.2777778	0.33333334	0.30555558
34	date	3	article	0.2777778	0.42857143	0.35317463

Figure 34: Calcul de la matrice des mesures de similarité finales

i.ligne	Concept.i	i.colonne	Concept c	Valeur Min	Valeur initiale	moyenne
0	foafperson	0	foafperson	1	1	1
3	article	0	foafperson	0.7	0.3	0.5
5	monograph	0	foafperson	0.7	0.6	0.65
11	inbook	0	foafperson	0.8	0.3	0.55
13	inproceedings	0	foafperson	0.7	0.4615385	0.5807692
19	misc	0	foafperson	0.7	0.1	0.4
20	proceedings	0	foafperson	0.7	0.5454546	0.6227273
26	journal	0	foafperson	0.7	0.4	0.55
27	conference	0	foafperson	0.25	0.5	0.375
28	address	0	foafperson	0.3055556	0.4	0.3527778
30	publisher	0	foafperson	0.7	0.4	0.55
32	personlist	0	foafperson	0.1176471	0.6	0.3588235
33	pagerange	0	foafperson	0.3055556	0.5	0.4027778
34	date	0	foafperson	0.3055556	0.2	0.2527778
0	foafperson	3	article	0.4444445	0.3	0.3722222
3	article	3	article	1	1	1
5	monograph	3	article	0.7	0.2222222	0.4611111
11	inbook	3	article	0.6	0.1428572	0.3714286
…						

Table 12: Calcul de la matrice des mesures de similarité finales

I.7 Détermination de seuil de similarité : Th=1.0

I.8 Concepts alignés : le programme filtre les déférentes valeurs des mesures de similarité finales calculées avec l'utilisation de seuil de similarité th, pour réduire le nombre de faux résultats :

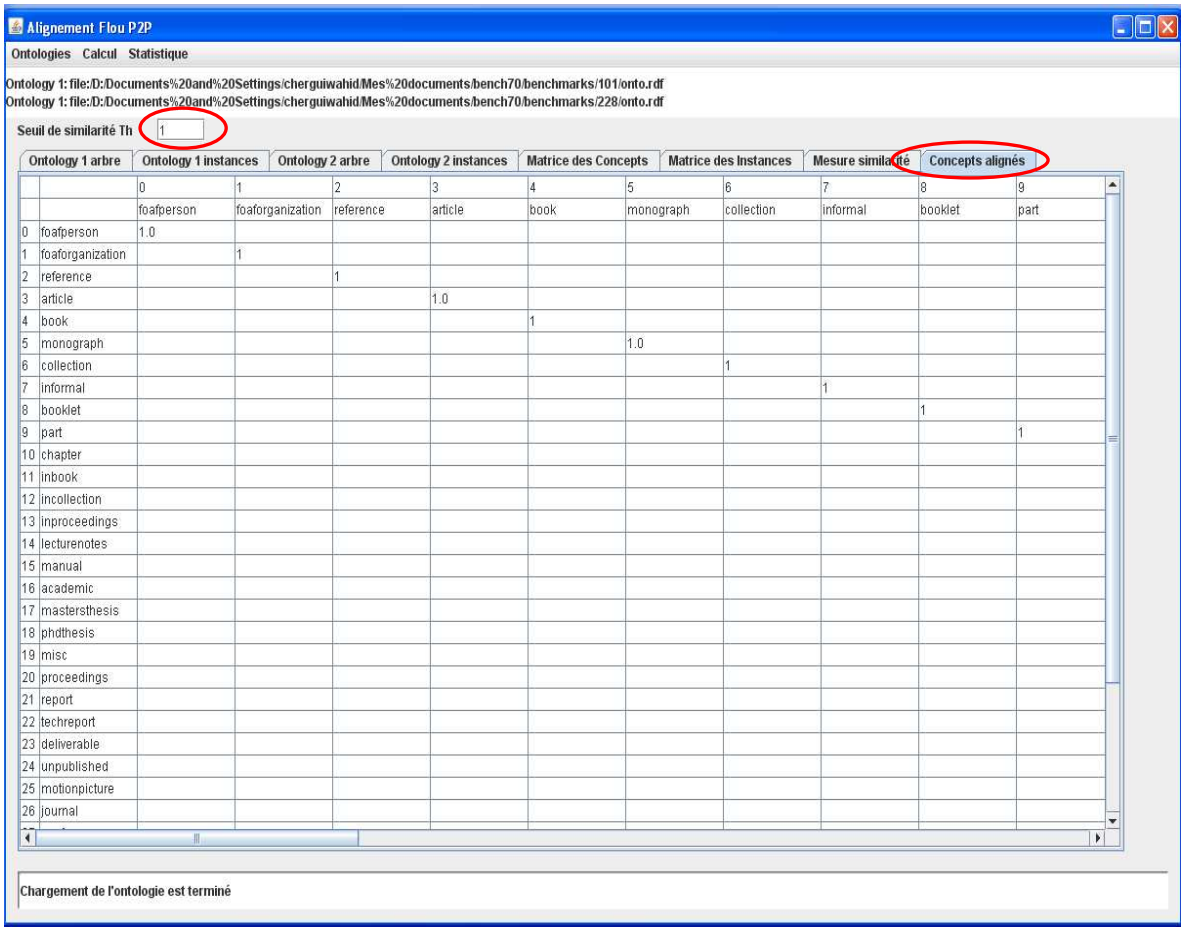


Figure 35: La matrice après le filtrage des résultats d'alignement

	Ordre	0	1	2	3	4	5	6
Ordre	**Concepts**	foafperson	foaforganization	reference	article	book	monograph	collection
0	foafperson	**1**						
1	foaforganization		**1**					
2	reference			**1**				
3	article				**1**			
4	book					**1**		
5	monograph						**1**	
6	collection							**1**
7	…							

Table 13 : La matrice après le filtrage des résultats d'alignement

I.9 Résultats et statistique : La fenêtre affiche le chemin des deux ontologies chargées et des informations à propos de chaque ontologie sont affichées : nombre des concepts, nombre des instances et le seuil de similarité th, La fenêtre affiche aussi les correspondances trouvées par l'algorithme et le nombre de ces correspondances.

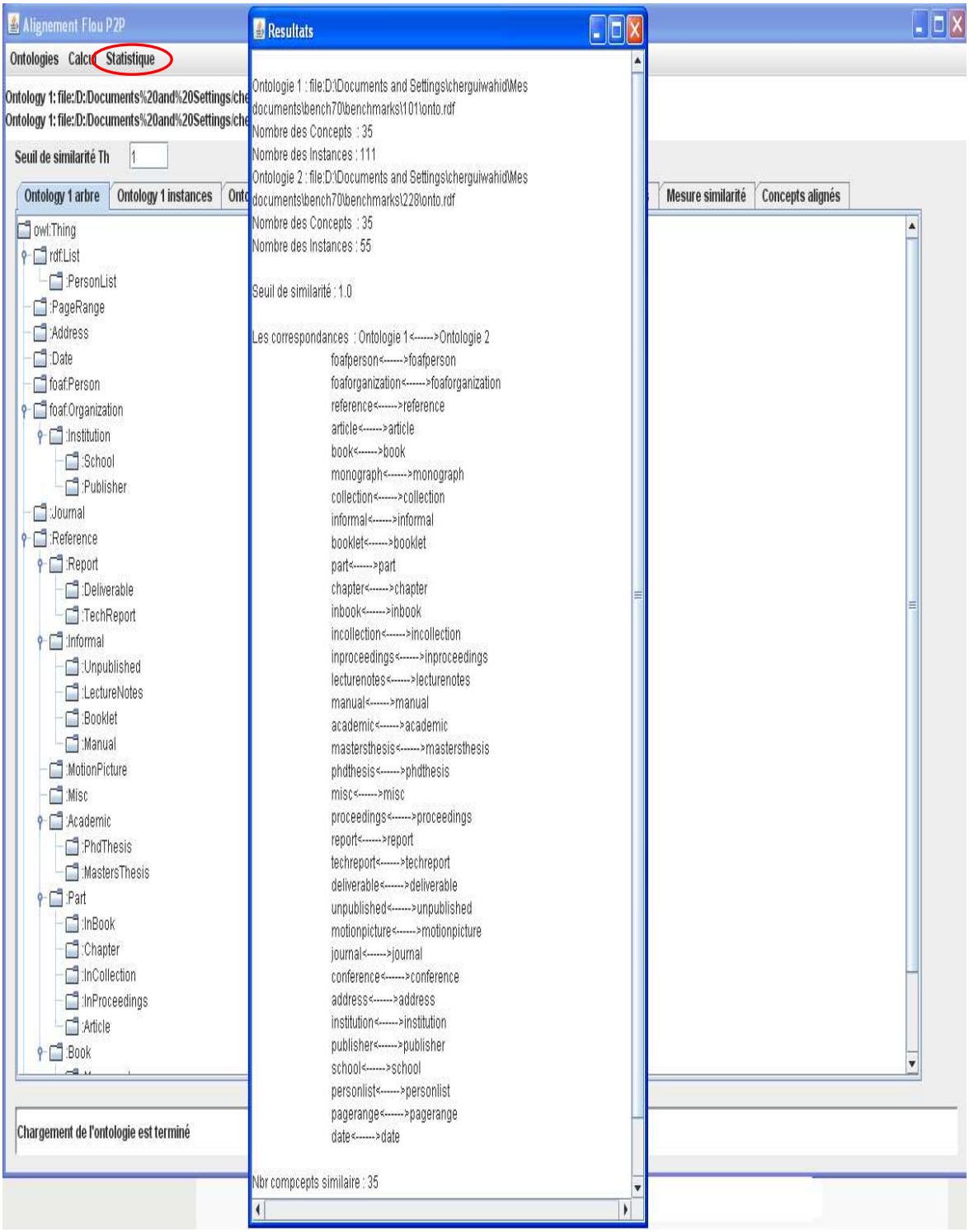


Figure 36: Résultats d'alignement produit

Ontologie 1 : file:D:\Documents and Settings\cherguiwahid\Mes documents\bench70\benchmarks\101\onto.rdf
Nombre des Concepts : 35
Nombre des Instances : 111
Ontologie 2 : file:D:\Documents and Settings\cherguiwahid\Mes documents\bench70\benchmarks\228\onto.rdf
Nombre des Concepts : 35
Nombre des Instances : 55

Seuil de similarité : 1.0

Les correspondances : Ontologie 1<------>Ontologie 2
foafperson<------>foafperson
foaforganization<------>foaforganization
reference<------>reference
article<------>article
book<------>book
monograph<------>monograph
.....
school<------>school
personlist<------>personlist
pagerange<------>pagerange
date<------>date

Nbr concepts similaires : 35

Table 14 : Résultats d'alignement produit

II. Evaluation du système:

Il existe deux catégories d'évaluation des systèmes d'alignement, selon le type de benchmark choisi : [Euzenat et al., 2007]

a. « les benchmarks de compétence » : composés de plusieurs unités de tests qui caractérisent une situation définie et permettent de mettre en évidence les aptitudes d'un système d'alignement en particulier ;

b. « les benchmarks de performance ou les benchmarks de comparaison» : permettent de comparer les performances de différents systèmes d'alignement sur la base de situations du monde réel.

Notre évaluation est dans le cadre du 1er type, à savoir : Les benchmarks de compétence.

II.1 Le format d'OAEI[1]:

C'est un format utilisé dans la campagne de test OAEI (Ontology Alignment Evaluation Initiative). Cette campagne fournit des paires d'ontologies à aligner, accompagnant avec les correspondances correctes entre deux ontologies dans les fichiers de référence dans leurs propres formats. Notre algorithme permet de faire aligner ces paires d'ontologies dans cette campagne et utilise leur fichiers de référence fournis pour évaluer la performance de l'algorithme.

II.2 Les métriques utilisées pour l'évaluation :

Afin de bien comprendre les résultats expérimentaux de notre algorithme « alignontoflou », une définition des métriques d'évaluation est nécessaire.

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

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

La *précision* est la proportion des correspondances correctes parmi l'ensemble de celles renvoyées par l'algorithme. Cette mesure reflète la *précision* d'un algorithme. Plus la valeur de *précision* est élevée, plus le bruit dans le résultat de l'algorithme est réduit, et donc plus la qualité de résultat est imposante (Figure 37).

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

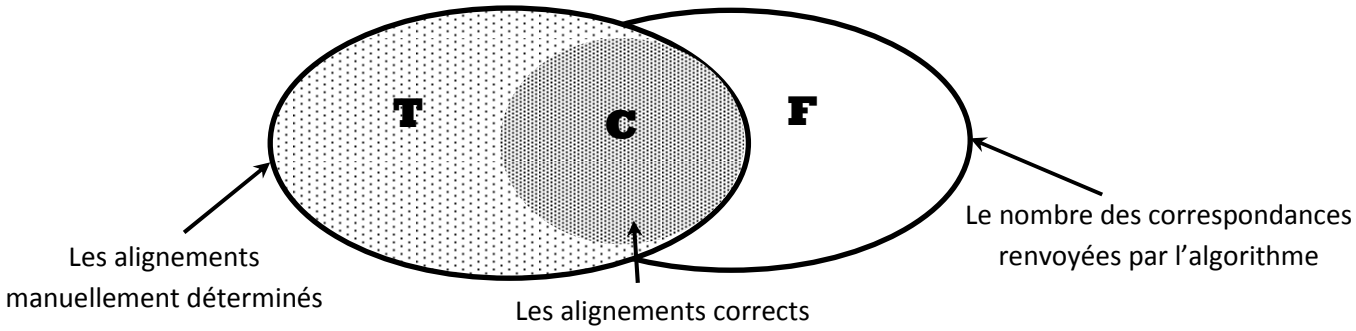


Figure 37: Présentation graphique des métriques d'évaluation

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

$$F_{mesure} = \frac{2 * rappel * précision}{rappel + précision}$$

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

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

Nous notons :

- ✓ F : Le nombre des correspondances renvoyées par l'algorithme.
- ✓ T : Le nombre des correspondances correctes déterminées manuellement par des experts (celles dans le fichier de référence).
- ✓ C : Le nombre des correspondances correctes trouvées.
- ✓ I = F – C : Nombre des correspondances incorrectes trouvées.
- ✓ M = T - C : Nombre des correspondances correctes mais pas trouvées.

Ainsi,

$$précision = \frac{C}{F}$$

$$rappel = \frac{C}{T}$$

$$F_{mesure} = \frac{2 * R * P}{R + P}$$

$$overall = R * \left(2 - \frac{1}{P}\right)$$

II.3 Évaluation des paires d'ontologies dans la campagne OAEI :

Nous avons testé l'algorithme proposé avec les paires d'ontologies de test dans la campagne: OAEI. Pour chaque paire d'ontologie, notre algorithme est exécuté avec un seuil de similarité th=1.

La campagne de tests de OAEI 2010 se compose de suite d'ontologies de tests. Les tests dans cette suite sont générés à partir d'une ontologie de référence (#101) dans le domaine de la *bibliographie*.

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

Nous avons testé l'algorithme d'alignement proposé sur les paires d'ontologies de test dans cette campagne, et comparé les résultats obtenus avec ceux des participants publiés à http://oaei.ontologymatching.org/2010/results/.

La table 15 résume les valeurs du *rappel*, la *précision* et la F_{mesure} pour chaque test, ainsi que celles de toute la campagne. Ces valeurs sont représentées dans la figure 38 :

Tests	*Précision*	*Rappel*	F_{mesure}	*Overall*
101	0.9375	0.3092	0,4650	0,2886
203	1	0.3298	0,4960	0,3298
208	1	0,0103	0,0204	0,0103
221	0.9354	0.2989	0,4530	0,2783
222	0.9032	0.2886	0,4374	0,2577
223	0.9354	0.3092	0,4648	0,2878
228	0.8857	0.9393	0,9117	0,8181
230	0.9259	0.3472	0,5050	0,3194
231	0.9375	0.3092	0,4650	0,2886
232	0.9062	0.2989	0,4495	0,2680
233	0.8285	0.8787	0,8529	0,6968

Table 15 : Valeur de *Précision* et *Rappel* et F_{mesure} et *Overall* pour chaque test

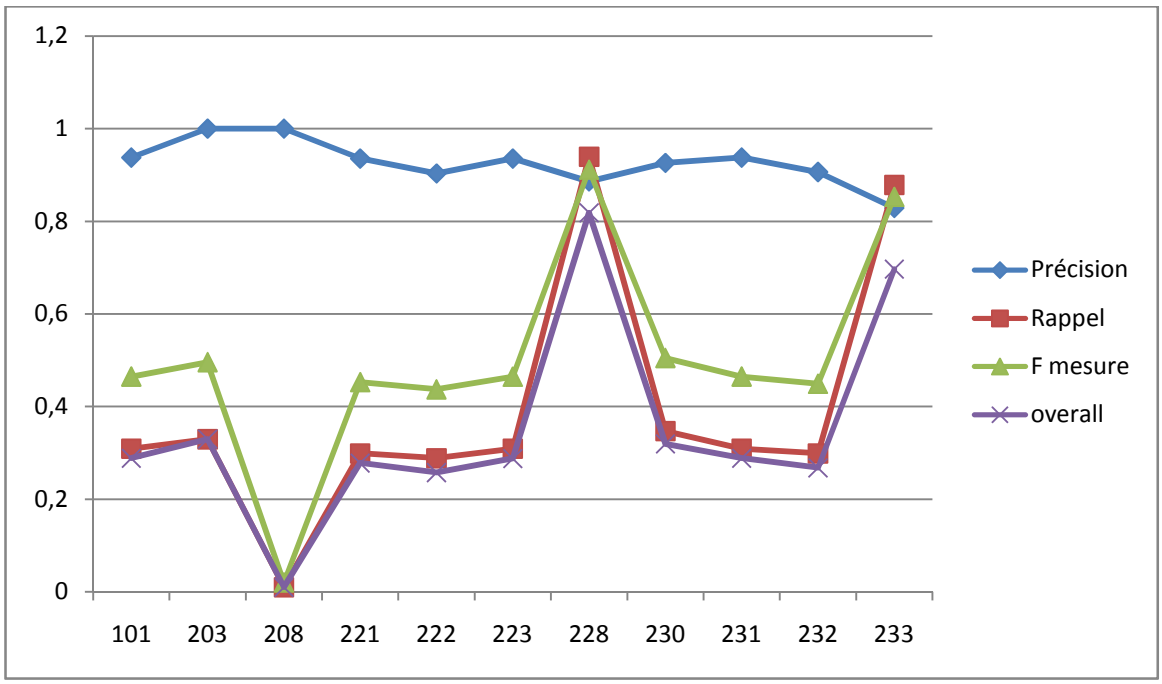


Figure 38 : Résultats de l'algorithme sur les paires d'ontologies de test

La Figure 38 montre le *rappel*, la *précision*, la F_{mesure} et *overall* obtenus par notre algorithme «alignontoflou» tout au long du processus avec 11 tests de référence (11 ontologies de test dans la campagne OAEI).

Pour chaque paire d'ontologie, notre algorithme «alignontoflou» est exécuté, Le seuil est utilisé dans l'algorithme pour déterminer des correspondances considérées comme correctes. La valeur de similarité finale de deux concepts de deux ontologies, calculée par des mesures terminologiques et extensionnelles, est comparée avec le seuil. Deux concepts sont considérées comme similaires si leur valeur de similarité dépasse ce seuil. Le résultat final de l'algorithme (la liste des correspondances entre deux ontologies) dépend donc du seuil choisi.

Plus le seuil de similarité est élevé, plus la qualité du résultat est élevée (la précision augmente, le nombre des correspondances incorrectes trouvées diminue) mais plus des correspondances correctes sont considérées comme incorrectes.

En considère que le seuil de similarité th=1, les valeurs de F_{mesure} et d'overall diminuent car le nombre des correspondances correctes mais pas trouvées devient très important. La raison est qu'il y a des entités qui représentent des concepts similaires mais leurs définitions dans des ontologies sont un peu différentes, donc leur valeur de similarité, calculée en se basant sur leurs définitions, n'atteint pas une valeur élevé (le maximum est 1.0), et alors ne dépasse pas le seuil.

Les correspondances trouvées par l'algorithme sont comparées avec les correspondances correctes dans le fichier de référence (fourni aussi par les campagnes de test), pour calculer le nombre des correspondances correctes et incorrectes trouvées par l'algorithme. Les valeurs des mesures d'évaluation (*précision, rappel*, F_{mesure}*, overall*) sont calculées pour chaque test et affichées dans un tableau et un graphique. L'algorithme «alignontoflou» cherche des correspondances entre deux ontologies par le calcul des mesures de similarités floues simples et combinées, On voit que le matcher flou produit une précision qui peut prendre des valeurs importantes aux dépens du rappel, en retournant que les correspondances correctes cependant peu nombreuses.

III. Conclusion

Dans ce chapitre, nous avons présenté l'implémentation de notre algorithme d'alignement d'ontologies. Nous avons le conçu avec NetBeans IDE version 6.9.1, qui permet de montrer la technique utilisée pour l'alignement des ontologies.

Nous avons réalisé cet algorithme en 6 étapes décrites comme suit :

Etape 1 : chargement des ontologies : Niveau Interface

Etape 2 : Validation des ontologies (avec Pellet) avec extraction des concepts et des instances de chaque ontologie.

Etape 3 : Cette étape est appelée l'étape de normalisation au niveau linguistique: avant de calculer les valeurs de similarité entre les concepts, les chaines de caractères (des concepts) sont nettoyées, pour que les résultats de la comparaison des chaines seront améliorés.

Etape 4 : Calcul des mesures de confiances simples et combinées pour chaque concept et instance (présentées dans le chapitre précédent), dans une première matrice qui contient les concepts des deux ontologies chargées ainsi une deuxième matrice qui contient les instances.

Etape 5: Détermination du seuil de similarité Th entre deux concepts, Th est utilisé pour filtrer les résultats d'alignement des concepts et pour écarter les résultats d'alignement avec une valeur de similarité très basse.

Etape 6 : l'algorithme retourne les valeurs de similarité du mapping entre le concept source et les concepts cibles, ainsi que les correspondances produites qui peuvent être stocké dans l'espace de stockage des alignements.

Nous avons présenté aussi l'évaluation de l'algorithme d'alignement d'ontologies que nous avons proposées et présentées précédemment. L'implémentation des algorithmes a été effectuée en Java et les évaluations ont été faites sur des bases de tests standards publiés à http://oaei.ontologymatching. org/.

Conclusion et Perspectives :

I. Conclusion Générale :

Cette thèse investit le domaine de l'alignement des ontologies dans l'environnement pair à pair. Dans ce contexte, des techniques sont proposées afin d'accomplir la tâche de l'alignement.

Pour les besoins de notre travail, nous avons élaboré un état de l'art sur les domaines des ontologies, de l'alignement des ontologies et sur le paradigme pair à pair, puis, nous avons présenté l'algorithme qui permet de mettre en œuvre une stratégie de partage de correspondances entre les pairs, et de leur permettre de collaborer pour découvrir des correspondances en utilisant la connaissance dont il dispose chaque pair. Pour cela, Étant donné un pair source et un pair cible, cet algorithme calcule l'alignement des concepts parmi les concepts contenus dans les ontologies. Il adopte une stratégie d'alignement basée sur l'exploitation des mesures de similarité floue simples ($\mu^{d,d'}$, $\mu^{a,a'}$, μ_{nomcon}) et des mesures de similarité floue combinées (μ_{dom} ,μ_{con})

Notre objectif est de proposer une solution qui se divise en deux parties :

a) la coordination entre les pairs dans un environnement pair à pair :

Dans cette étape un pair source envoi une requête RQ = <c_s> a tous les pairs du réseau, cette requête contient un concept c_s de l'ontologie source Os. Cette demande atteint les pairs cibles qui exécutent l'algorithme à son niveau. Le module d'agrégation (ou de combinaison) de l'algorithme exploite la fonction de similarité par la combinaison des valeurs de similarité fourni par les comparateurs et filtre les résultats en fonction du Th (le seuil de similarité). Pius, l'algorithme retourne les valeurs de similarité des correspondances entre le concept source et les concepts cibles au pair source.

b) L'algorithme d'alignement :

Qui s'exécute au niveau de chaque pair dans cet environnement. Nous avons réalisé cet algorithme en 6 étapes décrites comme suit :

Etape 1 : chargement des ontologies : Niveau Interface

Etape 2 : Validation des ontologies (avec Pellet) avec extraction des concepts et des instances de chaque ontologie.

Etape 3 : Cette étape est appelée l'étape de normalisation (le niveau linguistique) : avant de calculer les valeurs de similarité entre les concepts, les chaines de caractères (des concepts) sont nettoyées, pour que les résultats de la comparaison des chaines seront améliorés.

Etape 4 : Calcul des mesures de confiances simples et combinées pour chaque concept et instance (présentées dans le chapitre précédent), dans une première matrice qui contient les concepts des deux ontologies chargées ainsi une deuxième matrice qui contient les instances.

Cette étape se compose de deux phases :

i- Calcul des mesures de similarité de base : Sont les mesures de similarité basée sur les instances numériques $\mu^{d,d\prime}$ (en fonction de leurs distances euclidiennes) et non numériques $\mu^{a,a'}$ (Les mesures de similarité des chaines de caractère), et les mesures de confiance basée sur le nom du concept $\mu_{nomcon}^{A,A\prime}$.

ii- Calcul des mesures de similarité de domaine μ_{dom} qui combinent les mesures de similarité de l'ensemble des instances.

Les mesures de confiance combinée pour le mapping des concepts est déterminée par la combinaison ou l'agrégation des mesures de confiance basée sur les noms des concepts (μ_{nomcon}) et les mesures de confiance de domaine (μ_{dom}).

Si cette valeur finale est plus grande qu'un seuil prédéfini, les deux concepts sont considérés similaires.

Etape 5: Détermination du seuil de similarité Th : Th est utilisé pour filtrer les résultats d'alignement des concepts (comme c'est le cas dans la plupart des travaux existants) et pour écarter les résultats d'alignement avec une valeur de similarité très basse et pour réduire le nombre de fausses correspondances.

Etape 6: l'algorithme retourne les valeurs de similarité du mapping entre le concept source et les concepts cibles.

Les correspondances trouvées par l'algorithme sont comparées avec les correspondances correctes dans le fichier de référence (fourni par la campagne de test OAEI), pour calculer le nombre des correspondances correctes et incorrectes trouvées par l'algorithme. Les valeurs des mesures d'évaluation (*précision*, *rappel*, F_{mesure}, *overall*) sont calculées pour chaque cas et affichées dans un tableau et un graphique.

II. Perspectives :

Le caractère extensible de notre solution fait que nous pouvons ajouter :

- De nouvelles méthodes de calcul de similarité à tout moment (Intégration de nouveaux matchers) pour améliorer les résultats.
- Des ressources auxiliaires telles que les dictionnaires de synonymes et d'hyponymes comme WordNet (des méthodes lexicales), qui effectue la correspondance à travers les relations lexicales (par exemple, synonymie, hyponymie, etc.)
- De matcher au niveau structurel qui compare les structures internes des entités (par exemple, intervalle de valeur, cardinalité d'attributs, etc. De nombreux travaux existants relatifs au problème de matching de schémas utilisent ce type de matcher pour lequel l'appariement entre les attributs de schémas se fait sur leur type de données…

Bibliographie

[Baader et al., 2003] Baader, F., Calvanese, D., McGuinness, D., Nardi, D. et Patel-Schneider, P. (éditeurs), *The Description Logic Handbook : Theory, Implementation and Applications*. Cambridge University Press, pp. 495505.

[Bach Than Le, 2006] Bach Thanh Le, *Construction d'un Web sémantique multi-points de vue*, Thèse de doctorat Informatique, Ecole des Mines de Paris à Sophia Antipolis, 23 octobre 2006.

[Bachimont, 2000] Bruno Bachimont. *Engagement Sémantique et Engagement Ontologique: Conceptionet Réalisation D'ontologies En Ingénierie Des Connaissances*, chapter 19, pages 305–324. Eyrolles, 2000.

[Benetti et al., 2002] Benetti I., Beneventano D., Bergamaschi S., Guerra F., Vincini M. *An Information Integration Framework for e-commerce*. IEEE Intelligent Systems, 2002.

[Bergamaschi et al., 2004] Sonia Bergamaschi, Pablo R. Fillottrani, Gionata Gelati. *The SEWASIE Multi-agent System*. Proc of third International Workshop on Agents and Peer-to-Peer Computing (AP2PC 2004). Pages 120-131. 2004.

[Bouquet et al., 2003] Bouquet, P., Giunchiglia, F., van Harmelen, F., Serafini, L., et Stuckenschmidt, H. *C-OWL : contextualizing ontologies*. International Semantic Web Conference 2003: 164-179

[Charlet et al., 2004] Charlet J., Bachimont B., Troncy R., *Ontologies pour le Web Sémantique, Revue I3, numéro Hors Série «Web sémantique»*p. 43-63, 2004.

[Chen, 1999] Weiqin Chen, Riichiro Mizoguchi. Communication Content Ontology For Learner Model Agent in multi-Agent Architecture. July 19-23th, 1999. Workshop on Ontologies for Intelligent Educational Systems, Ninth International Conference on Artificial Intelligence in Education, AI-ED'99, Le Mans, France.http://www.ei.sanken.osaka-u.ac.jp/aied99/aied99-onto.html

[Cox, 1994] Cox, T. et Cox, M. *Multidimensional Scaling*. Chapman and Hall, 1994.

[Dagan et al., 1999] Dagan, I., Lee, L., et Pereira, F. *Similarity-based models of word cooccurrence probabilities*. Machine Learning 34(1-3).

[Do et al., 2002] Do, H., Melnik, S., Rahm, E. (2002). *Comparison of schema matching evaluations*, Proceedings of the 2nd Int. Workshop on Web Databasess, German Informatics Society, Erfurt, Germany. pp. 221-237

[Do et Rahm, 2002] Do, H.H. et Rahm, E. *COMA: a system for flexible combination of schema matching approaches*. Dans Proc. VLDB, pages 610–621, 2002.

[Doan et al., 2003] Doan, A.H., Madhavan, J., Dhamankar, R., Domingos, P. et Halevy, A. *learning to match ontologies on the semantic web*. VLDB journal, 2003.

[Durban et al., 1998] Durban, R., Eddy, S. R., Krogh, A., et Mitchison, G. *biological sequence analysis – Probabilistic models of proteins and nucleic acid*. Cambridge University Press, 1998.

[Ehrig et Staab, 2004] Ehrig, M. et Staab, S. *QOM: Quick Ontology Mapping*. Dans Proc. 3rd ISWC, Hiroshima (JP), November 2004.

[Ehrig et Sure, 2004] Ehrig, M. et Sure, Y. *Ontology Mapping: an integrated approach*. Dans Christoph Bussler, John Davis, Dieter Fensel, and Rudi Studer, editors, Proc. 1st ESWS, Hersounisous (GR), volume 3053 of Lecture Notes in Computer Science, pages 76–91. Springer Verlag, MAY 2004.

[Euzenat et al., 2007] Euzenat J., Ehrig M., Jentzsch A., Mochol M., Shvaiko P., *Case-based recommendation of matching tools and techniques, deliverable 1.2.2.2.1, Knowledge Web, Mars 2007.*

[Euzenat et Shvaiko, 2007] Jérôme Euzenat, Pavel Shvaiko. *Ontology Matching. 2007.*

[Euzenat et al., 2004] Euzenat, J., T. Le Bach, J. Barrasa, P. Bouquet, J. De Bo, R. Dieng et al. (2004). *D2.2.3: State of the art on ontology alignment*. Knowledge Web project, realizing the semantic web, EU-IST-2004-507482.

[Euzenat et Valtchev, 2004] Euzenat, J. et Valtchev, P. *Similarity-based ontology alignment in OWL-lite.* Dans Proc. 15th ECAI, Valencia (ES), 2004.

[Fagin et Wimmers, 1997] Fagin R, Wimmers E (1997). *Incorporating user preferences in multimedia queries*. In: Lecture notes in computer science, vol 1186.Springer, Berlin Heidelberg NewYork, pp 247–261

[Fagin, 1999] Fagin R (1999). *Combining fuzzy information from multiple systems*.J Comput Sys Sci 58:83–99

[Fellegi et Sunter, 1969] Fellegi, I. P. et Sunter, A. B. *A theory for record linkage*. Journal of the American Statistical Society 64:1183–1210.

[Fensel et al., 2007] Dieter Fensel, Holger Lausen, Axel Polleres, Jos de Bruijn, Michael Stollberg, Dumitru Roman, et John Domingue. *Enabling semantic web services: the web service modeling ontology*. Springer, Heidelberg (DE), 2007.

[Fürst, 2002] Fürst F. (2002). *L'ingénierie Ontologique* (02-07). Nantes: Institut de Recherche en Informatique de Nantes.

[Gal et al., 2005] A. Gal, A. Trombetta, A. Anaby-Tavor, D.Montesi, *A model for schema integration in heterogeneous databases*, In Proc. IDEAS Conference, Hong Kong, China, July 2003.

[Giunchiglia et al., 2004] Giunchiglia, F., Shvaiko, P. et Yatskevich, M. S-*Match: an algorithm and an implementation of semantic matching*. Dans Proceedings of ESWS 2004, Heraklion (GR), pages 61–75, 2004.

[Giunchiglia et Shvaiko, 2003] Giunchiglia, F. et Shvaiko, P. *Semantic Matching*. Dans Proc. IJCAI 2003 Workshop on ontologies and distributed systems, Acapulco (MX), pages 139–146, 2003.

[Giunchiglia et Yatskevich, 2004] Giunchiglia F. et Yatskevich M. *Element Level Semantic Matching*. dans Meaning Coordination and Negotiation workshop at ISWC'04. Hiroshima, Japan, 2004.

[Gomez Perez et Benjamins, 1999] Gomez Perez A., Benjamins V.R. *Overview of knowledge sharing and components: Ontologies and problem Solving Methods*. Workshop on Ontologies and problem-Solving Methods » 1999. Stockholm (Suède).

[Gomez-Perez, 1999] Gomez-Pérez A. (1999). *Tutorial on Ontological Engineering*. IJCAI.

[Gruber, 1993] Thomas Gruber. *A translation approach to portable ontology specifications. Knowledge Acquisition*, vol. 5, n″ 2, p. 199-220, 1993.

[Gruninger, 1995] Gruninger M. & Fox M. S. *Methodology for the design and evaluation of ontologies, in Proceedings of the Workshop on Basic Ontological Issues on Knowledge Sharing.* Montreal, 1995.

[Guarino et Giaretta, 1995] Nicola Guarino et Pierdaniele Giaretta. *Ontologies and knowledge bases: Towards a terminological clarification*. In N MARS, réd., Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing, pages 25–32. IOS Press, 1995.

[Guarino, 1997] Guarino N. (1997). *Some organizing principles for a unified top-level ontology*. *AAAI Spring Symposium on Ontological Engineering*, 57-63.

[He et al., 2003] He, B., Chang, K., & Han, J. (2003, December*). Automatic Complex Schema Matching across Web Query Interfaces: A Correlation Mining Approach*. Technical Report UIUCDCS-R-2003-2388, Department of Computer Science, UIUC.

[Ives et al., 2004] Zachary G. Ives, Alon Y. Halevy, Peter Mork, Igor Tatarinov: *Piazza: mediation and integration infrastructure for Semantic Web data*. Journal of Web Semantic, Volume 1 issue 2. Pages155-175. 2004.

[Jaro, 1989] Jaro, M. A. *Advances in record-linkage methodology as applied to matching the 1985 census of Tampa*, Florida. Journal of the American Statistical Association 84:414–420.

[Jaro, 1995] Jaro, M. A. *Probabilistic linkage of large public health data files* (disc: P687-689). Statistics in Medicine 14:491–498.

[Jiang et Conrath, 1997] Jiang, J., Conrath, D.: Semantic similarity based on corpus statistics and lexical taxonomy. In: Proc. of ROCLING X, Taiwan (1997)

[Kalfoglou et Schorlemmer, 2003] Kalfoglou, Y. et Schorlemmer, M. *If-map : an ontology mapping method base on information flow theory*. Journal of data semantics, 1:98–127, 2003.

[Klein et al., 2001] Klein, M., Gomez-Pérez, A., Gruninger, M., Stuckenschmidt, H. (2001, August 4-5). Combining and relating ontologies : an analysis of problems and solutions. Workshop on Ontologies and Information Sharing (IJCAI-01).

[Klir et Yuan, 1995] Klir GJ, Yuan B (eds). Fuzzy sets and fuzzy logic.Prentice-Hall, Englewood Cliffs, NJ.

[Li et Clifton, 2000] Li, W.S. et Clifton, C. *Semint: a tool for identifying attribute correspondences in heterogeneous database using neural networks*. Data Knowl.Eng., 33(1):49– 84, 2000.

[Lin, 1998] Dekang Lin. *An information-theoretic definition of similarity*. In Proc. 15th International Conference of Machine Learning (ICML), pages 296–304, Madison (WI US), 1998.

[Lovin, 1968] Lovins, J.B. *Development of a stemming algorithm. Mechanical Translation and Computational Linguistics*, 11: 22-31, 1968.

[Mädche et Staab, 2002] Mädche, A. et Staab, S. *Meauring similarity between ontologies*. In Proc. Of the 13th Int. Conference on Knowledge Engineering and Management (EKAW-2002), Siguenza, Spain, October 2002. Springer-Verlag.

[Madhavan et al., 2001] Madhavan, J., Bernstein, P. et Rahm, E. *Generic schema matching with cupid.* Dans Proceedings of the 27th International Conference on Very Large Data Bases, pages 49–58. Morgan Kaufmann Publishers Inc., 2001.

[Maynard et Ananiadou, 1999] Maynard, D.G. et Ananiadou, S. *Term extraction using a similarity-based approach*. In Recent Advances in Computational Terminology. John Benjamins, 1999.

[Miller, 1995] Miller, A.G. *WordNet: A lexical database for English.* Communications of the ACM, 38(11):39–41, 1995.

[Milojicic et al., 2002] Milojicic et al, 2002] Dejan S. Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins, et Zhichen Xu. *Peer-to-Peer Computing.* Technical Report HPL-2002-57, HP Laboratories Palo Alto, March 2002. http://www.hpl.hp.com/techreports/2002/HPL-2002-57R1.html.

[Mizoguchi, 1998] Mizoguchi R. *A Step towards Ontological Engineering*. Juin, 1998. Paper presented at the 12th National Conference on AI of JSAI.

[Monge et Elkan, 1996] Monge, A., et Elkan, C. *The field-matching problem: algorithm and application*. In Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, 1996.

[Namyoun et al, 2006] Choi Namyoun, Song Il-yeol et Han Hyoil. *A survey on ontology mapping.* SIGMOD Rec., 35(3):34–41, September 2006.

[Neches et al., 1991] Robert Neches, Richard Fikes, Tim Finin, Thomas Gruber, Ramesh Patil, Ted Senator, and William R. Swartout. *Enabling Technology For Knowledge Sharing. AI Magazine, Volume 12, No. 3, Pages 37-56. Fall 1991.*

[Nejdl et al., 2003] W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. Schlosser, I. Brunkhorst, et A. Lser. *Super-peer-based routing and clustering strategies for RDF-based peer-to-peer networks*. Proceedings of the 12th World Wide Web Conference, ACM, 2003.

[Nejdl et al., 2002] W. Nejdl, B. Wolf, C. Qu, S. Decker, M. S. A. Naeve, M. Nilsson, M. Palmer, et T. Risch. *EDUTELLA: A P2P networking infrastructure based on RDF*. Proceedings of the 11th international conference on World Wide, ACM, 2002.

[Noy et al., 2000] Noy, N., Fergerson, R. W., Musen, M. A. (2000). *The knowledge model of Protégé2000: combining interoperability and flexibility*, in Proceedings of the International Conference on Knowledge Engineering and Knowledge Management (EKAW'00).

[Noy et Musen, 2000] Noy, N. F., Musen, M. A. (2000). *Prompt: algorithm and tool for automated ontology merging and alignment*. In Proceeding of Seventeenth National Conference on Artificial Intelligence AAAI.

[Noy et Musen, 2001] Noy, N. et Musen, M. *Anchor-Prompt : using non-local context for semantic matching* .Dans Proc. IJCAI 2001 workshop on ontology and information sharing, Seattle (WA US), pages 63–70, 2001.

[Noy, 2004] Natalya Noy. *Semantic integration: a survey of ontology-based approaches*. SIGMOD Rec., 33(4):65–70, 2004.

[Oram, 2001] Oram A. (2001). *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly.

[Pirrò et al., 2008] Giuseppe Pirrò, Massimo Ruffolo, Domenico Talia: An Algorithm for Discovering Ontology Mappings in P2P Systems. KES (2) 2008: 631-641

[Porter, 1980] Porter, M.F. *An Algorithm for Suffix Stripping*, Program, 14(3): 130-137, 1980.

[Psyché et al., 2003] Valéry Psyché, Olavo Mendes et Jacqueline Bourdeau. *Apport de l'ingénierie ontologique aux environnements de formation à distance*. Revue STICEF, 10, 2003.

[Rahm et Bernstein, 2001] Rahm, E. et Bernstein, P. *A survey of approaches to automatic schema matching*. VLDB Journal, 10(4):334–350, 2001.

[Ratnasamy et al., 2001] S. Ratnasamy, P. Francis, M. Handley, R. Karp, et S. Shenker. *A scalable content-addressable network*. In Proc. of ACM SIGCOMM'01, San Diego, CA, USA, August 2001.

[Resnik, 1999] Phillip Resnik. *Semantic similarity in a taxonomy: an information-based measure and its application to problems of ambiguity in natural language*. Journal of Artificial Intelligence Research, 11:95–130, 1999.

[Schneider et al., 2001] Peter Patel-Schneider, Tim Berners-Lee, Dan Brickley, Dan Connolly, Mike Dean, Stefan Decker, Dieter Fensel, Richard Fikes, Pat Hayes, Jeff Heflin, Jim Hendler, Ora Lassila, Deb cGuinness, Lynn Andrea, Stein Ian Horrocks et Frank van Harmelen. *Specification DAML+OIL*, 2001. http://www.daml.org/2001/03/daml+oil-index.html.

[Shirky, 2000] Shirky Clay. *What is P2P…And what isn't.* www.openp2p.com/pub/a/p2p/2000/11/24/shirky1-whatisp2p.html.

[Sowa, 2000] John F. Sowa . Ontology, Metadata, and Semiotics. In proceedings of the International Conference on Conceptual Structures, ICCS'2000 » 14 août 2000 Darmstadt, Allemagne. http://www.bestweb.net/~sowa/peirce/ontometa.html

[Storia et al., 2001] I. Storia, R. Morris, M. Kaashoek, et H. Balakrishnan. *Chord: A scalable peer-to-peer lookup service for Internet applications*. In Proc. of ACM SIGCOMM'01, San Diego, CA, USA, August 2001.

[Studer, 1998] Studer R., Benjamins R., (1998). *Knowledge Engineering: Principles and Methods.* Data Knowledge Engineering.

[Troncy et Issac ,2002] Troncy, R., Issac, A. (2002). *DOE: Une mise en œuvre d'une méthode de structuration différentielle pour les ontologies*, in Actes des journées francophones d'Ingénierie des Connaissances (IC'2002), pages 63-74.

[Uschold et Grüninger, 1996] Mike Uschold et Michael Grüninger. *Ontologies: principles, methods, and applications*. Knowledge Engineering Review, 11(2):93–155, 1996.

[Valtchev et Euzenat, 1997] Valtchev, P et Euzenat, J.; *Dissimilarity measure for collections of objects and value*. In P. Coen X. Liu et M. Berthold, editors, Proc. 2nd Symposium on Intelligent Data Analysis., volume 1280, pages 259–272, 1997.

[Valtchev, 1999] Valtchev, P. *Construction automatique de taxonomies pour l'aide à la représentation de connaissances par objets*. Thèse d'informatique, Université Grenoble 1, 1999.

[Winkler, 1999] Winkler, W. E. *The state of record linkage and current research problems*. Statistics of Income Division, Internal Revenue Service Publication R99/04. http://www.census.gov/srd/www/byname.html. 1999.

[Xu et al., 2003] Xu, L., et Embley, D. (2003, October 20). *Using Domain Ontologies to Discover Direct and Indirect Matches for Schema Elements*. In: Second International Semantic Web Conference (ISWC- 03).

[Yang et Garcia-Molina, 2003] Beverly Yang et Hector Garcia-Molina. *Designing a Super-peer Network*, IEEE International Conference on Data Engineering, 2003.

[Zaihrayeu, 2006] Ilya Zaihrayeu. *Towards Peer-to-Peer Information Management Systems*. PhD thesis, International Doctorate School in Information and Communication Technology, University of Trento, Trento (IT), March 2006.

[Zhihong et Mingtian, 2003] Zuo Zhihong et Zhou Mingtian. *Web ontology language owl and its description logic foundation*. pages 157 – 160. Parallel and Distributed Computing, Applications and Technologies, PDCAT' 2003., 2003.

Webiographie:

[DOE, 2002] Differential Ontology Editor Home Page, http ://opales.ina.fr/public.

[Gnutella, 2000] Gnutella site web. www.gnutella.com.

[OntoEdit, 2004] OntoEdit (2004). Ontology Editor Home Page, http ://www.ontoprise.de/com/.

[Protégé, 2000] Protege2000 Ontology Editor Home Page, http://protege.stanford.edu/.

[Seti, 2001] SETI@home. http://setiathome.ssl.berkeley.edu/.

[W3Ca, 2004] *Recommendation W3C OWL*, 2004. http://www.w3.org/TR/owl-ref/

[W3Cb, 2004] *Recommendation W3C RDF*, 2004. http://www.w3.org/TR/rdf-syntax-grammar/

[W3Cc, 2004] *Recommendation W3C RDF Schema*, 2004. http://www.w3.org/TR/rdf-schema/

[W3Cd, 2004] *Recommendation W3C XML*, 2004. http://www.w3.org/TR/2004/REC-xml-20040204/

[W3Ce, 2004] *Recommendation W3C XML Schema*, 2004. http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/

Annexe:
Programme principal:

```
package alignontoflou;

import java.awt.Color;
import java.awt.Component;
import java.util.Collections;
import java.util.HashSet;
import java.util.Iterator;
import java.util.Set;
import javax.swing.JTree;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.DefaultTreeCellRenderer;
import javax.swing.tree.DefaultTreeModel;
import javax.swing.tree.TreeCellRenderer;
import org.mindswap.pellet.jena.PelletInfGraph;
import org.mindswap.pellet.jena.PelletReasonerFactory;
import com.hp.hpl.jena.ontology.OntClass;
import com.hp.hpl.jena.ontology.OntModel;
import com.hp.hpl.jena.ontology.OntResource;
import com.hp.hpl.jena.rdf.model.ModelFactory;
import com.hp.hpl.jena.vocabulary.OWL;

public class ClassTree  {
   OntModel model;
   Set<OntResource> unsatConcepts;

   // render the classes using the prefixes from the model
   TreeCellRenderer treeCellRenderer = new DefaultTreeCellRenderer() {
   public Component getTreeCellRendererComponent( JTree tree, Object value, boolean sel,
                                       boolean expanded, boolean leaf, int row, boolean hasFocus )
{
         super.getTreeCellRendererComponent( tree, value, sel, expanded, leaf, row, hasFocus
);
         DefaultMutableTreeNode node = (DefaultMutableTreeNode) value;
         Set<OntResource> set = (Set) node.getUserObject();
         StringBuffer label = new StringBuffer();
         // a set may contain one or more elements
         if( set.size() > 1 )
            label.append( "[" );
         Iterator<OntResource> i = set.iterator();
         // get the first one and add it to the label
         OntResource first = i.next();
         label.append( model.shortForm( first.getURI() ) );
         // add the rest (if they exist)
```

```
        while( i.hasNext() ) {
            OntResource c = i.next();
            label.append( " = " );
            label.append( model.shortForm( c.getURI() ) );
        }
        if( set.size() > 1 )
            label.append( "]" );
        //afficher les noeud non valide en rouge
        if( unsatConcepts.contains( first ) ) {
            setForeground( Color.RED );
        }
        setText( label.toString() );
        setIcon( getDefaultClosedIcon() );
        return this;
    }
};

public ClassTree( String ontology ) throws Exception {
    // crier le  reasoner
    model = ModelFactory.createOntologyModel( PelletReasonerFactory.THE_SPEC );
    // create le  model de l'ontology
    System.out.print( "Lecture ..." );
     System.out.println( ontology );
    model.read( ontology );
    System.out.println( "Lecture terminer" );
    // chargement du  model pour le reasoner
    System.out.print( "Preparer..." );
    model.prepare();
    System.out.println( "terminer" );
    // classification et validation
    System.out.print( "Classification..." );
    ((PelletInfGraph) model.getGraph()).getKB().classify();
    System.out.println( "terminer" );
}

public Set<OntResource> collect( Iterator i ) {
    Set<OntResource> set = new HashSet<OntResource>();
    while( i.hasNext() ) {
        OntResource res = (OntResource) i.next();
        if( res.isAnon() )
            continue;
        set.add( res );
    }
    return set;
}

public JTree getJTree() {
    OntClass owlThing = model.getOntClass( OWL.Thing.getURI() );
    OntClass owlNothing = model.getOntClass( OWL.Nothing.getURI() );
    unsatConcepts = collect( owlNothing.listEquivalentClasses() );
```

```
        DefaultMutableTreeNode thing = createTree( owlThing );
        Iterator<OntResource> i = unsatConcepts.iterator();
        if( i.hasNext() ) {
           DefaultMutableTreeNode nothing = createSingletonNode( owlNothing );
           while( i.hasNext() ) {
              OntClass unsat = (OntClass) i.next();

              if( unsat.equals( owlNothing ) )
                 continue;
              DefaultMutableTreeNode node = createSingletonNode( unsat );
              nothing.add( node );
           }
        }
        // crier l'arbre
        JTree classTree = new JTree( new DefaultTreeModel( thing ) );
        classTree.setCellRenderer( treeCellRenderer );

         for( int r = 0; r < classTree.getRowCount(); r++ )
            classTree.expandRow( r );

        return classTree;
     }

null
     DefaultMutableTreeNode createTree( OntClass cls ) {
        if( unsatConcepts.contains( cls ) )
           return null;
        DefaultMutableTreeNode root = createNode( cls );
        Set processedSubs = new HashSet();
        // get only direct subclasses
        Iterator subs = cls.listSubClasses( true );
        while( subs.hasNext() ) {
           OntClass sub = (OntClass) subs.next();
           if( sub.isAnon() )
              continue;
           if( processedSubs.contains( sub ) )
              continue;
           DefaultMutableTreeNode node = createTree( sub );
           if( node != null ) {
              root.add( node );
              processedSubs.addAll( (Set) node.getUserObject() );
           }
        }
        return root;
     }

     DefaultMutableTreeNode createSingletonNode( OntResource cls ) {
        return new DefaultMutableTreeNode( Collections.singleton( cls ) );
     }
```

```
    DefaultMutableTreeNode createNode( OntClass cls ) {
        Set<OntResource> eqs = collect( cls.listEquivalentClasses() );

        return new DefaultMutableTreeNode( eqs );
    }
}
```

```
package alignontoflou;
import java.awt.event.ActionEvent;
import java.io.File;
import java.net.URI;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JFileChooser;
import javax.swing.filechooser.FileNameExtensionFilter;
import edu.stanford.smi.protegex.owl.model.OWLIndividual;
import edu.stanford.smi.protegex.owl.model.OWLModel;
import edu.stanford.smi.protegex.owl.model.OWLNamedClass;
import edu.stanford.smi.protegex.owl.ProtegeOWL;
import edu.stanford.smi.protegex.owl.model.RDFIndividual;
import edu.stanford.smi.protegex.owl.model.RDFSNamedClass;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;
import javax.swing.JEditorPane;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextArea;
import javax.swing.JTree;
import javax.swing.table.TableColumn;
import javax.swing.tree.DefaultMutableTreeNode;

public class Menupr extends javax.swing.JFrame {

    /** Creates new form Menupr */
    public Menupr() {
        initComponents();
jTextField1.setText("0.5");
cons=Float.parseFloat("0.5");
        jTree1= new JTree(new DefaultMutableTreeNode("Empty"));
  jScrollPane1.setViewportView(jTree1);
   jTree2= new JTree(new DefaultMutableTreeNode("Empty"));
  jScrollPane2.setViewportView(jTree2);    }

    private void initComponents() {
        labonto = new javax.swing.JLabel();
        bart = new javax.swing.JLabel();
        jTabbedPane1 = new javax.swing.JTabbedPane();
        jScrollPane1 = new javax.swing.JScrollPane();
        jTree1 = new javax.swing.JTree();
```

```
        jScrollPane5 = new javax.swing.JScrollPane();
        jScrollPane2 = new javax.swing.JScrollPane();
        jTree2 = new javax.swing.JTree();
        jScrollPane3 = new javax.swing.JScrollPane();
        jScrollPane6 = new javax.swing.JScrollPane();
        jScrollPane7 = new javax.swing.JScrollPane();
        jScrollPane8 = new javax.swing.JScrollPane();
        jScrollPane9 = new javax.swing.JScrollPane();
        jTextField1 = new javax.swing.JTextField();
        jLabel1 = new javax.swing.JLabel();
        jMenuBar1 = new javax.swing.JMenuBar();
        jMenu1 = new javax.swing.JMenu();
        jMenuItem1 = new javax.swing.JMenuItem();
        jMenuItem2 = new javax.swing.JMenuItem();
        jMenu2 = new javax.swing.JMenu();
        jMenuItem4 = new javax.swing.JMenuItem();
        jMenu3 = new javax.swing.JMenu();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("Alignement Flou P2P");
        setCursor(new java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));
        labonto.setBackground(new java.awt.Color(255, 255, 255));
        labonto.setOpaque(true);
        bart.setBackground(new java.awt.Color(255, 255, 255));
        bart.setBorder(new
javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.LOWERED));
        bart.setOpaque(true);
        jScrollPane1.setViewportView(jTree1);
        jTabbedPane1.addTab("Ontology 1 arbre", jScrollPane1);
        jTabbedPane1.addTab("Ontology 1 instances", jScrollPane5);
        jScrollPane2.setViewportView(jTree2);
        jTabbedPane1.addTab("Ontology 2 arbre", jScrollPane2);
        jTabbedPane1.addTab("Ontology 2 instances", jScrollPane3);

jScrollPane6.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL
_SCROLLBAR_ALWAYS);

jScrollPane6.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL_SCR
OLLBAR_ALWAYS);
        jScrollPane6.setAutoscrolls(true);
        jTabbedPane1.addTab("Matrice des Concepts", jScrollPane6);


jScrollPane7.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL
_SCROLLBAR_ALWAYS);

jScrollPane7.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL_SCR
OLLBAR_ALWAYS);
        jScrollPane7.setAutoscrolls(true);
        jTabbedPane1.addTab("Matrice des Instances", jScrollPane7);
```

```
jScrollPane8.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL
_SCROLLBAR_ALWAYS);

jScrollPane8.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL_SCR
OLLBAR_ALWAYS);
        jScrollPane8.setAutoscrolls(true);
        jTabbedPane1.addTab("Mesure similarité", jScrollPane8);


jScrollPane9.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL
_SCROLLBAR_ALWAYS);

jScrollPane9.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL_SCR
OLLBAR_ALWAYS);
        jScrollPane9.setAutoscrolls(true);
        jTabbedPane1.addTab("Concepts alignés", jScrollPane9);

        jTextField1.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseExited(java.awt.event.MouseEvent evt) {
                jTextField1MouseExited(evt);
            }
        });
        jTextField1.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyPressed(java.awt.event.KeyEvent evt) {
                jTextField1KeyPressed(evt);
            }
        });

        jLabel1.setText("Seuil de similarité Th");

        jMenu1.setActionCommand("Ontologies");
        jMenu1.setCursor(new java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));
        jMenu1.setLabel("Ontologies");

        jMenuItem1.setText("Ontologie 1");
        jMenuItem1.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mousePressed(java.awt.event.MouseEvent evt) {
                jMenuItem1MousePressed(evt);
            }
        });
        jMenu1.add(jMenuItem1);

        jMenuItem2.setText("Ontologie 2");
        jMenuItem2.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mousePressed(java.awt.event.MouseEvent evt) {
                jMenuItem2MousePressed(evt);
            }
        });
```

```
        jMenu1.add(jMenuItem2);
        jMenuBar1.add(jMenu1);
        jMenu2.setText("Calcul");
        jMenuItem4.setText("Mesures de similarités");
        jMenuItem4.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mousePressed(java.awt.event.MouseEvent evt) {
                jMenuItem4MousePressed(evt);
            }
        });
        jMenu2.add(jMenuItem4);
        jMenuBar1.add(jMenu2);
        jMenu3.setText("Statistique");
        jMenu3.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mousePressed(java.awt.event.MouseEvent evt) {
                jMenu3MousePressed(evt);
            }
        });
        jMenuBar1.add(jMenu3);
        setJMenuBar(jMenuBar1);
        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(labonto, javax.swing.GroupLayout.DEFAULT_SIZE, 1192,
Short.MAX_VALUE)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED_SIZE, 131,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jTextField1, javax.swing.GroupLayout.PREFERRED_SIZE, 49,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(998, Short.MAX_VALUE))
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(bart, javax.swing.GroupLayout.DEFAULT_SIZE, 1172,
Short.MAX_VALUE)
                .addContainerGap())
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jTabbedPane1, javax.swing.GroupLayout.PREFERRED_SIZE,
1133, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(49, Short.MAX_VALUE))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addComponent(labonto, javax.swing.GroupLayout.PREFERRED_SIZE, 45,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(1, 1, 1)
```

```
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED_SIZE, 17,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(jTextField1, javax.swing.GroupLayout.PREFERRED_SIZE, 20,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jTabbedPane1, javax.swing.GroupLayout.DEFAULT_SIZE, 632,
Short.MAX_VALUE)
            .addGap(18, 18, 18)
            .addComponent(bart, javax.swing.GroupLayout.PREFERRED_SIZE, 34,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addContainerGap())
        );

        pack();
    }// </editor-fold>
public static String replaceW(String text, String searchString, String replacementString) {
    StringBuffer sBuffer = new StringBuffer();
    int pos = 0;
    while ((pos = text.indexOf(searchString)) != -1) {
    sBuffer.append(text.substring(0, pos) + replacementString);
    text = text.substring(pos + searchString.length());
    }
    sBuffer.append(text);
    return sBuffer.toString();
    }

    private void jMenuItem1MousePressed(java.awt.event.MouseEvent evt) {

              try {
           // TODO add your handling code here:

 bart.setText(convertToMultiline("Chargement de l'ontologie en cours ........"));
             chem=" ";
             loadOntologyWithFileChooser(OntologyType.FIRST_ONTO_RDF);
             labonto.setText(convertToMultiline(chem));
           ClassTree Tree3;
           String OWLFile=chem1;
File OwlFile=new File(OWLFile);
URI OwlUri=OwlFile.toURI();
 System.out.println(OwlUri.toString());
           chem1="file:"+chem1;
           chem11=chem1;
           Tree3 = new ClassTree(chem1);
           jTree1=Tree3.getJTree();
           jScrollPane1.setViewportView(jTree1);
           jTree1.setVisible(true);
 bart.setText(convertToMultiline("Chargement de l'ontologie est terminé"));

 OWLModel owlModel = ProtegeOWL.createJenaOWLModelFromURI(OwlUri.toString());
```

```
 Collection classes = owlModel.getUserDefinedOWLNamedClasses();
descr.clear();
conc1.clear();
inst1.clear();
int ordre=0;
    //JTable tableau ;
   for(Iterator it = classes.iterator(); it.hasNext();){
      //OWLNamedClass cls = (OWLNamedClass) it.next();
       RDFSNamedClass cls = (RDFSNamedClass) it.next();
      Collection instances = cls.getInstances(false);
      String buf=cls.getBrowserText();
      buf=buf.toLowerCase();
      buf = buf.replaceAll(" ", "");
      buf = buf.replaceAll(",", "");
      buf = buf.replaceAll(";", "");
      buf = buf.replaceAll(":", "");
      buf = buf.replaceAll("!", "");
      buf = buf.replaceAll("/", "");
      buf = buf.replaceAll("[\\d]", "");
      buf=replaceW(buf,".","");
      buf=replaceW(buf,"(","");
      buf=replaceW(buf,")","");
      buf=replaceW(buf,"[","");
      buf=replaceW(buf,"]","");
      buf=replaceW(buf,"?","");
      buf=replaceW(buf,"_","");
      buf=replaceW(buf,"-","");
      buf=replaceW(buf,"@","");

//      System.out.println(buf);
      System.out.println("Class " + cls.getBrowserText()+ " (" + instances.size()+")");
      conc1.add(buf);
      for(Iterator jt = instances.iterator(); jt.hasNext();){
         //OWLIndividual individual = (OWLIndividual) jt.next();
         RDFIndividual individual = (RDFIndividual) jt.next();
         System.out.println(" - "+ individual.getBrowserText());
      /*Object[][] don =  {
                     {cls.getBrowserText(), individual.getBrowserText()}
                     };*/
       descr.add(ordre);
      descr.add(cls.getBrowserText());
      descr.add(individual.getBrowserText());
        String buf1=individual.getBrowserText();
      buf1=buf1.toLowerCase();
      buf1 = buf1.replaceAll(" ", "");
      buf1 = buf1.replaceAll(",", "");
      buf1 = buf1.replaceAll(";", "");
      buf1 = buf1.replaceAll(":", "");
      buf1 = buf1.replaceAll("!", "");
      buf1 = buf1.replaceAll("/", "");
```

```
        buf1=replaceW(buf1,".","");
        buf1=replaceW(buf1,"(","");
        buf1=replaceW(buf1,")","");
        buf1=replaceW(buf1,"[","");
        buf1=replaceW(buf1,"]","");
        buf1=replaceW(buf1,"?","");
        buf1=replaceW(buf1,"_","");
        buf1=replaceW(buf1,"-","");
        buf1=replaceW(buf1,"@","");
         inst1.add(ordre);
        inst1.add(buf1);
        }
        ordre++;
    }

 Object[][] don = new Object[(descr.size()/3)][3];
 int j=0;
          for(int i=0;i<descr.size();i=i+3){
                              don[j][0] = descr.get(i);
                            don[j][1] = descr.get(i+1);
                            don[j][2] = descr.get(i+2);
                            j=j+1;
                           }

         String  title[] = {"Ordre","Concept", "Instances"};
         JTable tableau = new JTable(don , title);
          tableau.setVisible(true);
          jScrollPane5.setViewportView(tableau);
        } catch (Exception ex) {
           Logger.getLogger(Menupr.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
    private void jMenuItem2MousePressed(java.awt.event.MouseEvent evt) {
        try {
       bart.setText(convertToMultiline("Chargement de l'ontologie en cours ........"));
           chem += "\n";
           loadOntologyWithFileChooser(OntologyType.SECOND_ONTO_RDF);
           labonto.setText(convertToMultiline(chem));
             ClassTree Tree3;
               String OWLFile=chem1;
File OwlFile=new File(OWLFile);
URI OwlUri=OwlFile.toURI();
 System.out.println(OwlUri.toString());
            chem1="file:"+chem1;
            chem22=chem1;
           Tree3 = new ClassTree(chem1);
           jTree2=Tree3.getJTree();
           jScrollPane2.setViewportView(jTree2);
           jTree2.setVisible(true);
 bart.setText(convertToMultiline("Chargement de l'ontologie est terminé"));
```

```
  OWLModel owlModel = ProtegeOWL.createJenaOWLModelFromURI(OwlUri.toString());
 Collection classes = owlModel.getUserDefinedOWLNamedClasses();
descr2.clear();
conc2.clear();
inst2.clear();
int ordre=0;
    //JTable tableau ;
   for(Iterator it = classes.iterator(); it.hasNext();){
      //OWLNamedClass cls = (OWLNamedClass) it.next();
       RDFSNamedClass cls = (RDFSNamedClass) it.next();
      Collection instances = cls.getInstances(false);
      String buf=cls.getBrowserText();
      buf=buf.toLowerCase();
      buf = buf.replaceAll(" ", "");
      buf = buf.replaceAll(",", "");
      buf = buf.replaceAll(";", "");
      buf = buf.replaceAll(":", "");
      buf = buf.replaceAll("!", "");
      buf = buf.replaceAll("/", "");
      buf = buf.replaceAll("[\\d]", "");
      buf=replaceW(buf,".","");
      buf=replaceW(buf,"(","");
      buf=replaceW(buf,")","");
      buf=replaceW(buf,"[","");
      buf=replaceW(buf,"]","");
      buf=replaceW(buf,"?","");
      buf=replaceW(buf,"_","");
      buf=replaceW(buf,"-","");
      buf=replaceW(buf,"@","");
      System.out.println("Class " + cls.getBrowserText()+ " (" + instances.size()+")");
      conc2.add(buf);
      for(Iterator jt = instances.iterator(); jt.hasNext();){
        //  OWLIndividual individual = (OWLIndividual) jt.next();
         RDFIndividual individual = (RDFIndividual) jt.next();
         System.out.println(" - "+ individual.getBrowserText());
         String buf1=(individual.getBrowserText());
      buf1=buf1.toLowerCase();
      buf1 = buf1.replaceAll(" ", "");
      buf1 = buf1.replaceAll(",", "");
      buf1 = buf1.replaceAll(";", "");
      buf1 = buf1.replaceAll(":", "");
      buf1 = buf1.replaceAll("!", "");
      buf1 = buf1.replaceAll("/", "");
      buf1=replaceW(buf1,".","");
      buf1=replaceW(buf1,"(","");
      buf1=replaceW(buf1,")","");
      buf1=replaceW(buf1,"[","");
      buf1=replaceW(buf1,"]","");
      buf1=replaceW(buf1,"?","");
      buf1=replaceW(buf1,"_","");
```

```
        buf1=replaceW(buf1,"-","");
        buf1=replaceW(buf1,"@","");
        descr2.add(ordre);
        descr2.add(cls.getBrowserText());
        descr2.add(individual.getBrowserText());
        inst2.add(ordre);
        inst2.add(buf1);
        }
        ordre++;
    }
Object[][] don = new Object[(descr2.size()/3)][3];

int j=0;
          for(int i=0;i<descr2.size();i=i+3){
                              don[j][0] = descr2.get(i);
                            don[j][1] = descr2.get(i+1);
                            don[j][2] = descr2.get(i+2);
                            j=j+1;
                           }

         String  title[] = {"Ordre","Concept", "Instances"};
         JTable tableau = new JTable(don , title);
          tableau.setVisible(true);
           jScrollPane3.setViewportView(tableau);

        } catch (Exception ex) {
           Logger.getLogger(Menupr.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    private void jMenuItem4MousePressed(java.awt.event.MouseEvent evt) {
Object[][] don1 = new Object[(conc1.size()+2)][(conc2.size()+2)];
int j=0;
          for(int i=0;i<conc1.size()+2;i=i+1){
             if (i<2)      don1[i][0] = "";
                else {
                              don1[i][0] = j;
                            don1[i][1] =conc1.get(j); //descr.get(i+1);
                            j=j+1;}
                           }
 j=0;
          for(int i=0;i<(conc2.size()+2);i=i+1){
             if (i<2) don1[0][i] = "";
             else {
                    don1[0][i] =j;
                    don1[1][i] = conc2.get(j);
                    j=j+1;
                  }
                     }
```

```
 String  title[]=new String [conc2.size()+2];
 j=0;
for(int i=0;i<conc2.size()+2;i=i+1){
        title[i]="";
}

    JTable tableau = new JTable(don1 , title);        // TODO add your handling code here:

      tableau.setVisible(true);
      TableColumn col;
            for(int i = 0; i < tableau.getColumnCount(); i++){
                  //On récupère le modèle de la colonne
          if (i!=0) {
                              col = tableau.getColumnModel().getColumn(i);
                              //On lui affecte la nouvelle valeur
                              col.setPreferredWidth(130);
                  col.setWidth(130);}
          else {
             col = tableau.getColumnModel().getColumn(i);
                              //On lui affecte la nouvelle valeur
                              col.setPreferredWidth(20);
                  col.setWidth(20);

                  }

             }
             for(int i = 0; i < tableau.getRowCount(); i++){
                  //On affecte la taille de la ligne à l'indice spécifié !
                  tableau.setRowHeight(i, 20);
             }
      String t1,t2 ;
for (int i=2; i < (tableau.getRowCount()); i++) {
              t1=String.valueOf(don1[i][1]) ;
              t1=t1.trim();
         for (int k=2; k < (tableau.getColumnCount()); k++) {
                t2=String.valueOf(don1[1][k]);
                t2=t2.trim();
               // System.out.println(t1+" "+t2);
                if (t1.equalsIgnoreCase(t2)) don1[i][k]=1;
                else
                {
                   int max=t1.length();
                   if (max<t2.length()) max=t2.length();
                   t1=t1.toLowerCase();
                   t2=t2.toLowerCase();
                   char mot1 [] ;
                   char mot2 [] ;
                   int res=0;
                   mot1 = t1.toCharArray() ;
                   mot2=t2.toCharArray() ;
```

```
                    //  System.out.println(String.copyValueOf(mot1)+"
"+String.copyValueOf(mot2));
                for (int l=0; l < (mot1.length); l++) {
                  int n=0;
                  while (n< mot2.length) {
                      if (mot1[l]==(mot2[n])) {
                      res++;
                      mot2[n]=' ';
                      n=(mot2.length)+1;
                           }
                      n++;
                  }
                }
                      String tt=mot2.toString();
                  don1[i][k]=(float)res/max;
                   //System.out.println(t1+" "+t2+" "+String.copyValueOf(mot2)+" "+
don1[i][k]+" "+res+" "+max+" ");
                }
               }
   }

//   la matrice des concept est calculer avec leur valeurs

// on commence à calculer la matrice des concepts

      Object[][] don2 = new Object[(inst1.size()/2)+2][(inst2.size()/2)+2];

     j=0;
 for(int i=0;i<(inst1.size()/2)+2;i=i+1){
                    if (i<2)    {  don2[i][0] = "";
                    System.out.println(don2[i][0]);}
                    else {
                          don2[i][0] = inst1.get(j);
                       don2[i][1] =inst1.get(j+1); //descr.get(i+1);
                       j=j+2;
                       }
                    }
   j=0;
      for(int i=0;i<(inst2.size()/2)+2;i=i+1){
           if (i<2) don2[0][i] = "";
           else {
                 don2[0][i] = inst2.get(j);
                 don2[1][i] = inst2.get(j+1);
                 j=j+2;
               }
                                  }

    String  title2[]=new String [(inst2.size()/2)+2];
    j=0;
   for(int i=0;i<(inst2.size()/2)+2;i=i+1){
```

```
title2[i]="";
}

JTable tableau2 = new JTable(don2 , title2);        // TODO add your handling code here:

    tableau2.setVisible(true);
    TableColumn col2;
            for(int i = 0; i < tableau2.getColumnCount(); i++){
                    //On récupère le modèle de la colonne
        if (i!=0) {
                            col2 = tableau2.getColumnModel().getColumn(i);
                            //On lui affecte la nouvelle valeur
                            col2.setPreferredWidth(130);
                col2.setWidth(130);}
        else {
           col2 = tableau2.getColumnModel().getColumn(i);
                            //On lui affecte la nouvelle valeur
                            col2.setPreferredWidth(20);
                col2.setWidth(20);

                }
           }
           for(int i = 0; i < tableau2.getRowCount(); i++){
                    //On affecte la taille de la ligne à l'indice spécifié !
                    tableau2.setRowHeight(i, 20);
           }


    //   phase de calcule
       for (int i=2; i < (tableau2.getRowCount()); i++) {
           t1=String.valueOf(don2[i][1]) ;
           t1=t1.trim();
           boolean trou=true;
           try {
                    int mi = Integer.parseInt(t1);
                    trou=false;
                   // System.out.println("C'est un entier");
                }
           catch (Exception e) {
                //  System.out.println("Je ne suis pas un entier, et alors ca te derange ?");
                  trou=true;
                  }
      if (trou){
       for (int k=2; k < (tableau2.getColumnCount()); k++) {  //boucle 1
              t2=String.valueOf(don2[1][k]);
              t2=t2.trim();
                boolean trou2=true;
                try {
                    int mi = Integer.parseInt(t2);
                    trou2=false;
```

```
                    //  System.out.println("C'est un entier");
                      }
                    catch (Exception e) {
                                        trou2=true;
                      }

                if (trou2) {
                 if (trou) {
                    if (t1.equalsIgnoreCase(t2)) don2[i][k]=1;
                    else
                    {
                       int max=t1.length();
                       if (max<t2.length()) max=t2.length();
                       t1=t1.toLowerCase();
                       t2=t2.toLowerCase();
                       char mot1 [] ;
                       char mot2 [] ;
                       int res=0;
                       mot1 = t1.toCharArray() ;
                       mot2=t2.toCharArray() ;
                      //  System.out.println(String.copyValueOf(mot1)+"
"+String.copyValueOf(mot2));
                         for (int l=0; l < (mot1.length); l++) {
                            int n=0;
                                  while (n< mot2.length) {
                                  if (mot1[l]==(mot2[n])) {
                                  res++;
                                  mot2[n]=' ';
                                  n=(mot2.length)+1;
                                  }
                                  n++;
                                  }
                          }
                       // String tt=mot2.toString();
                       don2[i][k]=(float)res/max;
                   }
                   }
                    else
                    {
                    don2[i][k]=0; //comparaison texte avec entier=0
                    }
                    }
                } //fin boucle 1
                } //fin si comparaison texte
             else   // comparaison si element entier
             {
                    for (int k=2; k < (tableau2.getColumnCount()); k++) {  //boucle 2
                     t2=String.valueOf(don2[1][k]);
                     t2=t2.trim();
                       boolean trou2=true;
```

```
                try {
                   int mi = Integer.parseInt(t2);
                   trou2=false;
                   }
                catch (Exception e) {
                   trou2=true;
                   }
            if (trou2=false) {
             if (trou=false) {
                if (t1.equalsIgnoreCase(t2)) don2[i][k]=1;
                else
                {
                   int dif=Integer.parseInt(t1)-Integer.parseInt(t2);
                   int t3=Integer.parseInt(t1);
                   int t4=Integer.parseInt(t2);
                   if (t3>t4)
                  don2[i][k]=Integer.parseInt(t1)-Integer.parseInt(t2);
                   else
                      don2[i][k]=Integer.parseInt(t2)-Integer.parseInt(t1);
                 }
                 }
              else
                {
                don2[i][k]=0; //comparaison entier avec text=0
                }
                }
            } //fin boucle 2

        }
}

 ArrayList inst3 = new ArrayList();
 inst3.clear();
      j=2;

      for(int i = 2; i < tableau2.getColumnCount(); i++){

         if ((i+1)==(tableau2.getColumnCount()-1)) {
            t1=String.valueOf(don2[0][i]);
         t2=String.valueOf(don2[0][i+1]);
               if ((t1.compareTo(t2)!=0))  {
                  inst3.add(don2[0][i]);
                inst3.add(j);
                inst3.add(i);
                 j=i+1;

               }
            System.out.println(tableau2.getColumnCount()+" "+j+" "+i);
                inst3.add(don2[0][i+1]);
                inst3.add(j);
```

```
                    inst3.add(i+1);
                    j=i+1;
                    break;
                   }
             t1=String.valueOf(don2[0][i]);
             t2=String.valueOf(don2[0][i+1]);
                  if ((t1.compareTo(t2)!=0))  {
                    inst3.add(don2[0][i]);
                   inst3.add(j);
                   inst3.add(i);
                    j=i+1;
                  }

              }
      //    System.out.println(inst3);

Object[][] don3 = new Object[(inst3.size()/3)][5];
j=0;
          for(int i=0;i<(inst3.size()/3);i=i+1){
                              don3[i][0] = inst3.get(j);
                          don3[i][1] =inst3.get(j+1); //descr.get(i+1);
                          don3[i][2] =inst3.get(j+2);
                          j=j+3;
                      }
String title3[]=new String [5];
          for(int i = 0; i < 5; i++){
             title3[i]="";
          }

   ArrayList inst4 = new ArrayList();
    inst4.clear();
for (int l=0; l < ((inst3.size()/3)); l++) {
        for (int i=2; i < (tableau2.getRowCount()); i++) {

                t1=String.valueOf(don2[i][1]) ;
                t1=t1.trim();
                boolean trou3=true;
                try {
                        int mi = Integer.parseInt(t1);
                        trou3=false;
                    }
                catch (Exception e) {
                        trou3=true;
                    }

                 float max1=0;
              int val1 =Integer.parseInt(String.valueOf(don3[l][1]));
              int val2=Integer.parseInt(String.valueOf(don3[l][2]));
             if (val1==val2) {
                String tt=String.valueOf(don2[i][val1]);
```

```
            float max=Float.parseFloat(tt);
            max1=max;
            //System.out.println(max+" "+max1+" "+val1+" "+" "+val2+"   seul ");
          }

          for (int ii=val1; ii <= (val2); ii++) {
            String tt=String.valueOf(don2[i][ii]);
            float max=Float.parseFloat(tt);
            //if (val1>100) System.out.println(max+" "+max1+" "+val1+" "+" "+val2+"
"+ii+" "+i);
            if (max>max1) max1=max;
            //if (val1>100) System.out.println(max+" "+max1+" "+val1+" "+" "+val2+"
"+ii+" "+i);
          }// le max est calculé pour chaque position ligne fixe*colone varie de ii


           if (trou3=false){   // si est entier le max est connu on calcule 1-val/max
              if (val1==val2) {
                  String tt=String.valueOf(don2[i][val1]);
                  float max=Float.parseFloat(tt);
                   don2[i][val1]=(float)(1-(max/max1));
                   }
              for (int ii=val1; ii < val2; ii++) {
                   String tt=String.valueOf(don2[i][ii]);
                   float max=Float.parseFloat(tt);
                   don2[i][ii]=(float)(1-(max/max1));
                   }
                 max1=0;
              if (val1==val2) {
                   String tt=String.valueOf(don2[i][val1]);
                   float max=Float.parseFloat(tt);
                   max1=max;
                   }
              for (int ii=val1; ii <= val2; ii++) {
                   String tt=String.valueOf(don2[i][ii]);
                   float max=Float.parseFloat(tt);
                   if (max>max1) max1=max;
                   }
                System.out.println(" entier   ");
            }
            // System.out.println(l+" "+max1+" "+val1+" "+" "+val2+" ");
             don3[l][3]=max1;
             don3[l][4]=don2[i][0];
             inst4.add(don3[l][0]);
             inst4.add(don3[l][1]);
             inst4.add(don3[l][2]);
             inst4.add(max1);
             inst4.add(don2[i][0]);

    }
```

```
  }


    //// fin calcul matrice instance
    // on cherche le max puis min par ligne
    Object[][] don4 = new Object[(inst4.size()/5)][5];
    j=0;
          for(int i=0;i<(inst4.size()/5);i=i+1){
                          don4[i][0] =inst4.get(j);
                        don4[i][1] =inst4.get(j+1); //descr.get(i+1);
                        don4[i][2] =inst4.get(j+2);
                        don4[i][3] =inst4.get(j+3); //descr.get(i+1);
                        don4[i][4] =inst4.get(j+4);
                        j=j+5;  }
    JTable tableau3 = new JTable(don4 , title3);

           //tableau3.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
/*JTable tableau44 = new JTable(don4 , title3);
   tableau44.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);

jScrollPane7.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS
);

jScrollPane7.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_AL
WAYS);
   jScrollPane7.setAutoscrolls(true);
   jScrollPane7.setViewportView(tableau44);*/

  ArrayList inst5 = new ArrayList();
    inst5.clear();
         j=0;
         int lk=0;
         for(int i = 0; i < tableau3.getRowCount(); i++){

            if ((i+1)==(tableau3.getRowCount())) {
                 inst5.add(don4[i][4]);
                t2=String.valueOf(don4[i][4]);
                int t3=Integer.parseInt(t2);
                t3=t3+2;
                inst5.add(don1[t3][1]);
                inst5.add(don4[i][0]);
                 t1=String.valueOf(don4[i][0]);
                 int t4=Integer.parseInt(t1);
                 t4=t4+2;
                 inst5.add(don1[1][t4]);

              //     System.out.println(t3+" "+t4+" "+j);
                  float min1=1;
                for(int kk=j ; kk <(i+1); kk++){
                String tt=String.valueOf(don4[kk][3]);
```

```
        float min=Float.parseFloat(tt);
        if (min<min1) min1=min;
        }
        inst5.add(min1);
        inst5.add(don1[t3][t4]);
        String tt=String.valueOf(don1[t3][t4]);
        float min=Float.parseFloat(tt);
       float moy=0;
       if (min1>0) moy=(min+min1)/2;
       else moy=min;
        inst5.add(moy);
        j=i+1;
         break;
        }
     t1=String.valueOf(don4[i][4]);
t2=String.valueOf(don4[i+1][4]);
       if ((t1.compareTo(t2)!=0))  {

       inst5.add(don4[i][4]);
       t2=String.valueOf(don4[i][4]);
       int t3=Integer.parseInt(t2);
       t3=t3+2;
       inst5.add(don1[t3][1]);
       inst5.add(don4[i][0]);
        t1=String.valueOf(don4[i][0]);
        int t4=Integer.parseInt(t1);
        t4=t4+2;
        inst5.add(don1[1][t4]);

         float min1=1;
        for(int kk=j ; kk <(i+1); kk++){
        String tt=String.valueOf(don4[kk][3]);
        float min=Float.parseFloat(tt);
        if (min<min1) min1=min;
        }
        inst5.add(min1);
        inst5.add(don1[t3][t4]);
        String tt=String.valueOf(don1[t3][t4]);
        float min=Float.parseFloat(tt);
       float moy=0;
       if (min1>0)
       {
          if (min>0) moy = (min + min1) / 2;
          else moy=min1;
        }
       else  moy=min;
        inst5.add(moy);
        j=i+1;
        }
}
```

```
  Object[][] don5 = new Object[(inst5.size()/7)][7];
  String title5[]= {"i. ligne","Concept.l","i. colonne","Concept.C","valeur Minum","Valeur
Initial","Moyen"};
    j=0;
          for(int i=0;i<(inst5.size()/7);i=i+1){
                          don5[i][0] =inst5.get(j);
                        don5[i][1] =inst5.get(j+1); //descr.get(i+1);
                        don5[i][2] =inst5.get(j+2);
                        don5[i][3] =inst5.get(j+3); //descr.get(i+1);
                        don5[i][4] =inst5.get(j+4);
                         don5[i][5] =inst5.get(j+5);
                          don5[i][6] =inst5.get(j+6);
                        j=j+7;  }
    JTable tableau4 = new JTable(don5 , title5);

          // fin calcul de min par instance pour chaque concept


//setForeground(Color.getColor("red"));
tableau4.setVisible(true);
          TableColumn col3;
                for(int i = 0; i < tableau4.getColumnCount(); i++){
                if (i!=0) {
                                col3 = tableau4.getColumnModel().getColumn(i);
                           col3.setPreferredWidth(130);
                        col3.setWidth(130);}
              else {
                        col3 = tableau4.getColumnModel().getColumn(i);
                                col3.setPreferredWidth(130);
                        col3.setWidth(130);
                        }
                }
                for(int i = 0; i < tableau4.getRowCount(); i++){
                          tableau4.setRowHeight(i, 20);
                }

//System.out.println(String.valueOf(tableau3));

          for(int i = 0; i < tableau4.getRowCount(); i++){
                t1=String.valueOf(don5[i][0]);
                t2=String.valueOf(don5[i][2]);
                int t3=Integer.parseInt(t1);
                int t4=Integer.parseInt(t2);
                t3=t3+2;
                t4=t4+2;
                  String tt=String.valueOf(don5[i][6]);
                float min=Float.parseFloat(tt);
                don1[t3][t4]=min;
                //don1[t3][t4]=98;
          }
```

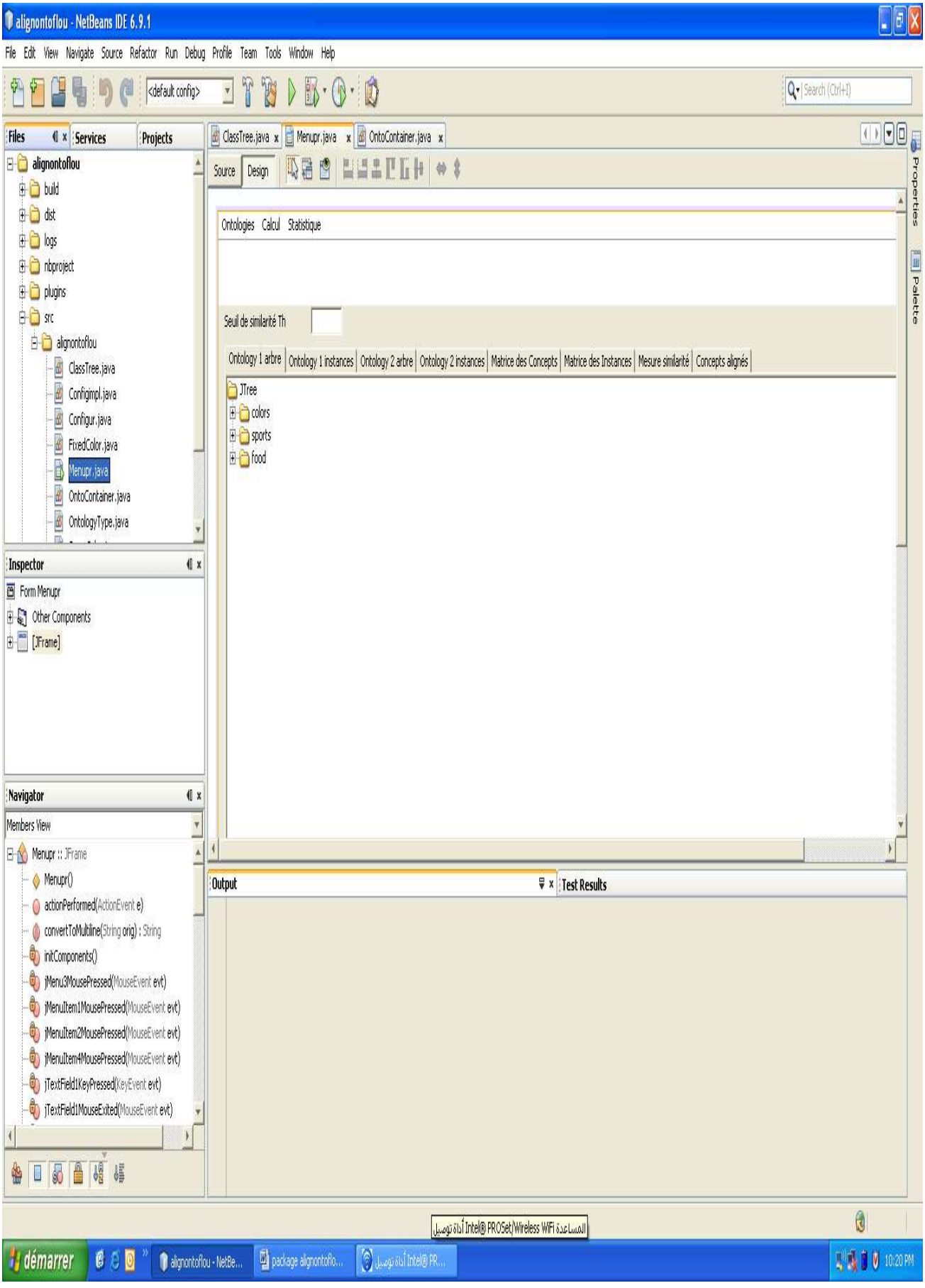
```
  Object[][] donf = new Object[(conc1.size()+2)][(conc2.size()+2)];
  j=0;
          for(int i=0;i<conc1.size()+2;i=i+1){
             if (i<2)      donf[i][0] = "";
                else {
                             donf[i][0] = j;
                           donf[i][1] =conc1.get(j); //descr.get(i+1);
                           j=j+1;}
                         }
  j=0;
          for(int i=0;i<(conc2.size()+2);i=i+1){
             if (i<2) donf[0][i] = "";
             else {
                   donf[0][i] =j;
                   donf[1][i] = conc2.get(j);
                   j=j+1;
                 }
                    }

  int ta1=(tableau.getRowCount()-2);
    int Ia1=(tableau2.getRowCount()-2);
  resul="";
  resul="Ontologie 1 : "+chem11+"\n";
  resul=resul+"Nombre des Concepts  : "+ta1+"\n";
resul=resul+"Nombre des Instances : "+Ia1+"\n";

  int ta2=(tableau.getColumnCount()-2);
    int Ia2=(tableau2.getColumnCount()-2);
resul=resul+"Ontologie 2 : "+chem22+"\n";
  resul=resul+"Nombre des Concepts  : "+ta2+"\n";
resul=resul+"Nombre des Instances : "+Ia2+"\n"+"\n";
resul=resul+"Seuil de similarité : "+cons+"\n"+"\n";
resul=resul+"Les correspondances  : "+"Ontologie 1<------>Ontologie 2"+"\n";
int compt=0;
          for(int i = 2; i < tableau.getRowCount(); i++){
             for( j = 2; j < tableau.getColumnCount(); j++){
                t2=String.valueOf(don1[i][j]);
                if (t2.length()>0){
                   float t3=Float.parseFloat(t2);
                if (t3>=cons){
                   t1=String.valueOf(don1[i][1]);
                   t2=String.valueOf(don1[1][j]);
                   resul=resul+"                              "+t1+"<------>"+t2+"\n";
                   donf[i][j]=don1[i][j];
                   compt++;
                }
                }
              }

          }
```

```
resul=resul+"\n"+ "Nombre de  concepts similaire  : "+compt+"\n"+"\n";
        JTable tableauf = new JTable(donf , title);        // TODO add your handling code here:
          tableauf.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);

          TableColumn colf;
                for(int i = 0; i < tableauf.getColumnCount(); i++){
                if (i!=0) {
                                colf = tableauf.getColumnModel().getColumn(i);
                   colf.setPreferredWidth(100);
                  colf.setWidth(130);}
              else {
                        colf = tableauf.getColumnModel().getColumn(i);
                                colf.setPreferredWidth(20);
                        colf.setWidth(20);
                       }
                }
                for(int i = 0; i < tableauf.getRowCount(); i++){
                        tableauf.setRowHeight(i, 20);
                }


jScrollPane8.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS
);

jScrollPane8.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_AL
WAYS);
   jScrollPane8.setAutoscrolls(true);
   jScrollPane8.setViewportView(tableau4);

   tableau2.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);

jScrollPane7.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS
);

jScrollPane7.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_AL
WAYS);
   jScrollPane7.setAutoscrolls(true);
   jScrollPane7.setViewportView(tableau2);

   tableau.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);

jScrollPane6.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS
);

jScrollPane6.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_AL
WAYS);
   jScrollPane6.setAutoscrolls(true);
   jScrollPane6.setViewportView(tableau);
   tableauf.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
```

```
jScrollPane9.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS
);

jScrollPane9.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_AL
WAYS);
   jScrollPane9.setAutoscrolls(true);
   jScrollPane9.setViewportView(tableauf);
   }

   private void jTextField1MouseExited(java.awt.event.MouseEvent evt) {
      // TODO add your handling code here:

      cons=Float.parseFloat(jTextField1.getText());

   }

   private void jMenu3MousePressed(java.awt.event.MouseEvent evt) {
JFrame f = new JFrame("Resultats ");
JScrollPane jsp=new JScrollPane();
   JTextArea ta = new JTextArea();
   //ta.setText("That's one small step for man...\nOne giant leap for mankind.");
   ta.setText("\n"+resul+"\n");
   ta.setLineWrap(true);
   ta.setWrapStyleWord(true);

   jsp.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
   jsp.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
   jsp.setAutoscrolls(true);
   jsp.setViewportView(ta);

   f.getContentPane().add(jsp);
   f.setSize(500, 500);
   f.setVisible(true);

   }

   private void jTextField1KeyPressed(java.awt.event.KeyEvent evt) {
     cons=Float.parseFloat(jTextField1.getText());
   }
    public void actionPerformed(ActionEvent e)
   {
      Object Source = e.getSource();
      if(Source == jMenuItem1)
      {
         loadOntologyWithFileChooser(OntologyType.FIRST_ONTO_OWL);
         System.out.println("selection du fichier 1 ");
      }
      if(Source == jMenuItem2)
      {
         loadOntologyWithFileChooser(OntologyType.SECOND_ONTO_OWL);
```

```
            System.out.println("selection du fichier 2");
        }
        if(Source == jMenu2)
        {
            System.out.println("source Quitter");
            dispose();
        }
    }

    private void loadOntologyWithFileChooser(final OntologyType ontoType)
    {
        JFileChooser jfilechooser = lastDirectory == null ? new JFileChooser() : new
JFileChooser(lastDirectory);
        FileNameExtensionFilter filenameextensionfilter =
ontoType.getOntoFormat().getFilter();
        jfilechooser.setFileFilter(filenameextensionfilter);
        int i = jfilechooser.showOpenDialog(null);
        if(i == 0)
        {
            File file = jfilechooser.getSelectedFile();

            bart.setText(convertToMultiline("Chargement de l'ontologie en cours ....."));
            lastDirectory = file.getParentFile();
          //  saveDynamicOnaguiInfos(lastDirectory);
           // configuration.setOntologyLastOpenDirectory(lastDirectory);
            final URI filename = file.toURI();
            //System.out.println(file);
            chem1="";
            chem1=file.toString();

            System.out.println((new StringBuilder()).append("Path to ontology:
").append(filename).toString());

            if (ontoType == OntologyType.SECOND_ONTO_OWL) {
            chem+=((new StringBuilder()).append("Ontology 2: ").append(filename).toString());
            }
            else
            {
                chem+=((new StringBuilder()).append("Ontology 1:
").append(filename).toString());
            }
            System.out.println(filename);
             bart.setText(convertToMultiline("Chargement de l'ontologie est terminé"));
            //System.out.println("Chargement de l'ontologie est terminé");
        }

    }

    public static String convertToMultiline(String orig)
{
```

```
    return "<html>" + orig.replaceAll("\n", "<br>");
}

    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new Menupr().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JLabel bart;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JMenu jMenu1;
    private javax.swing.JMenu jMenu2;
    private javax.swing.JMenu jMenu3;
    private javax.swing.JMenuBar jMenuBar1;
    private javax.swing.JMenuItem jMenuItem1;
    private javax.swing.JMenuItem jMenuItem2;
    private javax.swing.JMenuItem jMenuItem4;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JScrollPane jScrollPane2;
    private javax.swing.JScrollPane jScrollPane3;
    private javax.swing.JScrollPane jScrollPane5;
    private javax.swing.JScrollPane jScrollPane6;
    private javax.swing.JScrollPane jScrollPane7;
    private javax.swing.JScrollPane jScrollPane8;
    private javax.swing.JScrollPane jScrollPane9;
    private javax.swing.JTabbedPane jTabbedPane1;
    private javax.swing.JTextField jTextField1;
    private javax.swing.JTree jTree1;
    private javax.swing.JTree jTree2;
    private javax.swing.JLabel labonto;
  private File lastDirectory;
private javax.swing.JTable jTable1;
private String chem=" ";
private JEditorPane htmlPane;
private String chem1=" ";
private float cons=0;
private String chem11=" ";
private String chem22=" ";
private String resul="";
 ArrayList descr = new ArrayList();
  ArrayList descr2 = new ArrayList();
ArrayList conc1 = new ArrayList();
  ArrayList conc2 = new ArrayList();
  ArrayList inst1 = new ArrayList();
  ArrayList inst2 = new ArrayList();
}
```


alignontoflou - NetBeans IDE 6.9.1
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Files
Services
Projects
alignontoflou
build
dist
logs
nbproject
plugins
src
alignontoflou
ClassTree.java
Configimpl.java
Configur.java
FixedColor.java
Menupr.java
OntoContainer.java
OntologyType.java
ClassTree.java
Menupr.java
OntoContainer.java
Source
Design
Ontologies
Calcul
Statistique
Seuil de similarité Th
Ontology 1 arbre
Ontology 1 instances
Ontology 2 arbre
Ontology 2 instances
Matrice des Concepts
Matrice des Instances
Mesure similarité
Concepts alignés
JTree
colors
sports
food
Inspector
Form Menupr
Other Components
[JFrame]
Navigator
Members View
Menupr :: JFrame
Menupr()
actionPerformed(ActionEvent e)
convertToMultiline(String orig) : String
initComponents()
jMenu3MousePressed(MouseEvent evt)
jMenuItem1MousePressed(MouseEvent evt)
jMenuItem2MousePressed(MouseEvent evt)
jMenuItem4MousePressed(MouseEvent evt)
jTextField1KeyPressed(KeyEvent evt)
jTextField1MouseExited(MouseEvent evt)
Output
Test Results
Properties
Palette
démarrer
alignontoflou - NetBe...
package alignontoflo...
10:20 PM