Université de Montréal

Indigo: Une approche multi-stratégique et adaptative pour un alignement sémantique intégrant le contexte des données à apparier

par
Youssef Bououlid Idrissi

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Thèse présentée à la Faculté des études supérieures
en vue de l'obtention du grade de Philosophiæ Doctor (Ph.D.)
en informatique

Août, 2008

Université de Montréal
Faculté des études supérieures

Cette thèse intitulée:

Indigo: Une approche multi-stratégique et adaptative pour un alignement sémantique intégrant le contexte des données à apparier

présentée par:

Youssef Bououlid Idrissi

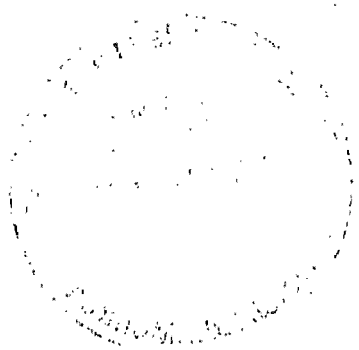
a été évaluée par un jury composé des personnes suivantes:

Houari Sahraoui
président de Jury

Iluju Kiringa
examinateur externe

Guy Lapalme
examinateur interne

Julie Vachon
directrice de recherche



Paul Arminjon
Représentant du doyen de la FESP

RÉSUMÉ

L'élaboration du matching sémantique entre sources de données hétérogènes est une étape fondamentale dans la conception d'applications pour le partage de données. En pratique, cette tâche est coûteuse et souvent exposée au risque d'erreur lorsqu'elle est exécutée manuellement. Par conséquent, plusieurs systèmes ont été développés pour son automatisation. Or, dans la majorité de cas, ces systèmes visent un type spécifique d'applications et se basent sur des approches rigides d'alignement. En effet, ils appliquent une même technique fixe de matching sur les données à aligner sans tenir compte de leurs caractéristiques sémantiques. Le problème est que de tels systèmes ne sont pas adaptés à la diversité d'Internet où les sources de données sont nombreuses et très hétérogènes, et évoluent constamment. Par ailleurs, la plupart des solutions actuelles mettent le focus seulement sur le problème d'alignement simple (*un-à-un*). Ceci est probablement dû au fait que le matching complexe (*plusieurs-à-plusieurs*) pose un problème beaucoup plus difficile à cause de l'espace de recherche des combinaisons possibles de concepts qui est très large.

Pour mieux faire face à l'hétérogénéité des sources de données, nous proposons INDIGO, un système capable d'élaborer un mapping sémantique en tenant compte du contexte des sources à aligner. INDIGO se distingue par sa faculté d'enrichir les sources de données par des informations sémantiques extraites de leur environnement de développement avant de procéder à leur alignement proprement dit. Par ailleurs, INDIGO s'appuie sur une approche d'alignement multi-stratégique et adaptative qui tient compte de la nature diversifiée des sources à apparier. Elle préconise l'utilisation de plusieurs stratégies de matching qui peuvent être combinées de façon dynamique en prenant en considération les spécificités sémantiques des concepts en cours de réconciliation.

Mots-clés : analyse de contexte, mapping complexe, alignement sémantique adaptatif, multi-stratégie.

ABSTRACT

The elaboration of semantic matching between heterogeneous data sources is a fundamental step in the design of data sharing applications. This task is tedious and often error prone if handled manually. Therefore, many systems have been developed for its automation. However, the majority of them target a specific type of applications and rely on rigid approaches applying, without distinction, the same matching technique with the same fixed procedures on data to be aligned without regard for their characteristics. The problem is that such mapping systems are not adapted to the diversity of Internet where data sources and domains are numerous, highly heterogeneous and often changing. Moreover, virtually all solutions currently focus on the problem of finding simple (*one-to-one*) matching. This is likely due to the fact that complex (*many-to-many*) matching raises a far more difficult problem since the search space of concept combinations can be tremendously large.

To better cope with data source heterogeneity, we propose INDIGO, a system which can compute simple and complex matching by taking into account data sources' context. The distinctive feature of INDIGO is to enrich data sources with semantic information extracted from their individual development artifacts. Thanks to this enrichment step, INDIGO can then compute a more accurate mapping between the two data sources thus enhanced. To generate the mapping, INDIGO relies on an adaptive multi-strategy matching approach which takes into account the diversified nature of data sources. It provides multiple matching strategies which can be flexibly combined to take into consideration the specificities of the data sources being aligned.

Key words : context analysis, complex semantic matching, adaptive semantic matching, multi-strategy.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

CHAPITRE 1

INTRODUCTION

Les exigences en interopérabilité ont pris une importance sans précédent suite à l'explosion du volume des communications transitant sur Internet, ce médium étant devenu le support d'échange privilégié tant des entreprises que des particuliers. Cette évolution a été soutenue entre autres, par l'adoption quasi-généralisée de XML comme format standard d'échange de données ainsi que par l'émergence de la technologie des services web facilitant le déploiement d'applications distribuées sur la toile Internet.

L'interopérabilité entre des acteurs différents est synonyme de possibilité d'intégration, de partage ou d'échange de leurs données. Toutefois, étant généralement développées isolément, les sources de données regroupées forment souvent des touts très hétérogènes. Autrement dit, chaque source peut se présenter sous un format syntaxique (e.g. XML, modèle relationnel, etc.) et sémantique différent. Ainsi, il n'est pas rare d'être aux prises avec deux concepts, issus de sources distinctes, ayant la même syntaxe mais ne portant pas le même sens. Dans ce contexte, l'interopérabilité implique forcément le développement de systèmes traducteurs pour assurer la mise en correspondance des différentes sources de données présentes sur le Web.

En pratique, cette mise en correspondance est une tâche très complexe. Que ce soit dans un but d'intégration ou de transformation des données, une étape cruciale d'alignement sémantique des concepts entre sources de données différentes, est dès lors nécessaire. Plus précisément, il s'agit de procéder à la mise en correspondance des concepts sources et cibles partageant une même sémantique mais qui peuvent se présenter sous des formes lexicales ou syntaxiques différentes. L'exécution manuelle de cette étape s'avère très laborieuse et sujette à de nombreux risques d'erreurs, surtout si le nombre de concepts à apparier devient important. Un groupe de travail de la compagnie de télécomunications GTE [1] avait entamé le processus d'intégration de 27000 éléments de données provenant de

[1] GTE (General Telephone and Electronics) était la plus large entreprise indépendante dans le domaine de la téléphonie fixe aux États-Unis pendant le temps du système Bell.

40 de ses applications [LC94]. L'expérience avait démontré qu'il avait fallu en moyenne 4 heures pour le traitement d'un seul élément de donnée s'il est exécuté par une personne externe au système d'information : le traitement incluait la découverte du matching et la documentation des éléments alignés. Les planificateurs du projet avaient estimé que dans ces conditions, le traitement intégral des données allait prendre plus que 12 années/homme comme charge de travail. Différentes solutions ont alors vu le jour pour tenter d'automatiser l'étape d'alignement sémantique des données.

1.1 Approches actuelles

Les solutions actuelles se divisent essentiellement en deux classes principales, chacune étant caractérisée par une vision particulière. La première défend l'importance de présenter explicitement la sémantique des concepts et de leurs relations au niveau des sources de données. Elle stipule qu'une description formelle de la sémantique, associée à des outils pour son analyse automatique est la meilleure voie à emprunter pour pallier l'hétérogénéité. Elle mise toutefois sur l'adoption unanime d'un certain langage de description des données par une très large communauté. Les partisans de cette première vision se sont donc lancés dans la création de standards sous forme de langages de description de sources de données sur Internet. Ceci correspond à la philosophie du Web Sémantique qui préconise une mutation progressive du Web classique lisible seulement par les humains vers un Web lisible à la fois par les machines et par les humains. Parmi les efforts les plus notables, nous pouvons citer ceux de l'organisation W3C visant la création du standard RDF (Ressource Description Framework) pour la description des ressources sur Internet ou le développement de langages de description ontologiques tels que OWL ou DAML+OIL. Le challenge à relever par cette première initiative se résume dans la réalisation du meilleur compromis entre, d'un côté, un degré élevé d'expressivité sémantique des langages de description ; et de l'autre côté, un degré peu élevé de complexité de la syntaxe de ces langages. Un niveau élevé d'expressivité de ces langages étant nécessaire pour tenir compte d'une marge de besoins la plus large possible. Et, un niveau peu élevé de leur complexité étant pour encourager leur adoption et permettre

des inférences simples et déterministes relatives aux sémantiques des concepts et de leurs relations.

Quant à la seconde vision, elle considère que la nature multiculturelle et très dynamique du Web constitue un obstacle permanent face à l'adoption de standards. Elle explore donc des solutions capables de s'adapter à l'évolution rapide et incessante de la diversité des ressources Web. Ces solutions se basent typiquement sur l'utilisation de techniques issues de l'intelligence artificielle pour élaborer le mapping sémantique entre des sources de données hétérogènes. Elles regroupent, entre autres, les systèmes à base de règles qui tablent essentiellement sur l'application d'heuristiques pour élaborer le matching entre concepts sources et cibles (e.g. un concept source de type réel ne peut pas correspondre à un concept cible de type date). On y trouve également les systèmes à apprentissage qui, une fois entraînés sur des ensembles de données exemples (e.g. alignement fourni manuellement entre sources de données pour l'entraînement) peuvent élaborer des modèles de classifications qu'ils utiliseront pour générer l'alignement entre de nouvelles sources. Il est toutefois à noter que les solutions adhérant à cette seconde vision restent bien entendu applicables pour l'alignement des sources de données soumises aux normes du Web sémantique puisqu'elles ne se limitent pas à supporter un format de données particulier. Elles disposeront même, dans ce cas, de plus d'informations sémantiques à exploiter, et pourront ainsi améliorer leur performance.

1.2 Notre solution : une approche multi-stratégique adaptative

Notre solution s'inscrit dans le cadre de la seconde vision et préconise plus particulièrement une approche multi-stratégique. L'idée principale derrière une telle approche repose sur le principe de combiner plusieurs stratégies (application des heuristiques, apprentissage, etc.) de rapprochement sémantique entre concepts afin d'augmenter la qualité de l'alignement final. Intuitivement, il se dégage très souvent une stratégie préférable pour le rapprochement de deux concepts étant donné un type d'informations considéré. Par exemple, si l'on considère les informations liées aux schémas de données (e.g. type de données, structure, etc.), l'application de règles heuristiques donnera pro-

bablement de meilleurs résultats que l'application de techniques d'apprentissage. En effet, étant donné le nombre limité de cas possibles de matchings dans ce cas particulier, il est possible d'utiliser d'une stratégie de mapping basée sur un ensemble de règles ciblées de mise en correspondance sémantique (e.g. entre des types de données sources et cibles) qui serait plus efficace. Inversement, les techniques d'apprentissage peuvent mieux se prêter au traitement du contenu de champs de données textuelles sachant que des expériences [DDH03] ont confirmé leur efficacité dans l'alignement basé sur la comparaison de contenus de concepts de ce type. Ainsi, on peut espérer qu'une combinaison pondérée de ces stratégies aidera forcément à améliorer la précision de l'alignement final. D'ailleurs, les approches multi-stratégiques gagnent de plus en plus en popularité comme en témoigne le nombre croissant des systèmes qui les adoptent et qui sont présentés dans la littérature [DDH03, JPE01, DR02]. Par exemple, le système **LSD** décrit dans [DDH03] repose exclusivement sur une approche d'apprentissage multi-stratégique. Il utilise plusieurs apprenants de base indépendants qui sont supervisés par un apprenant central appelé méta-apprenant. Chaque apprenant de base se spécialise dans le traitement d'un type spécifique d'informations puisées des schémas ou instances de sources de données. De là, chacun génère son propre modèle de classification. C'est le méta-apprenant qui est responsable de la combinaison finale des résultats d'alignement renvoyés individuellement par chaque apprenant de base.

Ces systèmes présentent des avantages majeurs notamment en terme d'adaptabilité et d'extensibilité : au besoin, une nouvelle stratégie peut être définie pour répondre aux contraintes spécifiques des sources de données particulières à aligner et être facilement combinée avec les autres stratégies existantes.

Néanmoins, ces systèmes souffrent encore des mêmes faiblesses que leurs antécédants mono-stratégiques ou hybrides (i.e. systèmes combinant deux ou plusieurs stratégies de façon rigide au sein d'un même algorithme donné). Ces limitations peuvent se résumer comme suit :

1. Tout d'abord, les informations puisées au sein des sources de données en vue de recouvrer la sémantique des concepts qui s'y trouvent, se limitent seulement aux schémas et instances de données. Or, souvent en pratique, ces informations

présentent des lacunes empêchant de les analyser correctement (e.g. nom de concept incomplet, ambigü ou abrégé)

2. Ensuite, dans la majorité de cas, le focus est mis seulement sur un alignement simple des données (mise en correspondance 1 à 1) alors que les alignements complexes sont fréquents en réalité (mise en correspondance d'une combinaison de concepts sources avec une combinaison de concepts cibles : $adresse \rightarrow concat(rue, ville)$)
3. Enfin, la combinaison des stratégies se fait souvent de façon rigide ne tenant pas compte des spécificités des sources de données à aligner. Par conséquent, la flexibilité additionnelle que devrait apporter une solution multi-stratégique est absente et on n'obtient alors rien de plus qu'avec un système hybride.

Ainsi, les objectifs que nous nous sommes fixés pour la présente thèse se résument en deux axes principaux :

1) Intégration du contexte externe d'une source de données : Nous avons décidé d'étendre et de diversifier les ressources utilisées pour l'extraction de la sémantique des données à aligner en intégrant ce que nous avons appelé le `contexte externe d'une source de données`. En effet, nous pensons que le fait d'isoler une représentation de données de son contexte (e.g. en ne considérant que son schéma par exemple) réduit d'avance le lot d'informations cognitives pouvant être exploitées et qui se cache pourtant derrière la conceptualisation de cette représentation. En pratique, le contexte (i.e. documents de conception d'une application, modèle de données décrit en `XMI`, contraintes d'intégrité des données exprimées en `OCL`, formulaires, programmes, etc.) dans lequel s'inscrit une source de données constitue un gisement informationnel très précieux pouvant être utilisé pour enrichir les sources de données à aligner. Il mérite d'être plus systématiquement exploré pour mieux cerner la sémantique des concepts à apparier. Il peut aussi être utilisé pour identifier des combinaisons sémantiques de concepts (e.g. $concat(rue, code\_postal)$) qui peuvent elles aussi participer au processus d'alignement et permettre par conséquent la génération d'un matching complexe.

2) Implémentation d'un système d'alignement adaptatif à approche multi-

stratégique : Nous entendons ici par système adaptatif, un système qui soit capable de sélectionner dynamiquement les stratégies individuelles à appliquer en fonction de la nature des informations sémantiques dont il dispose pour chaque concept à aligner. Pour assurer l'adaptativité, nous avons misé sur la flexibilité de combinaison des stratégies d'alignement qu'offre une architecture multi-stratégique. En effet, dans une telle architecture chaque stratégie est supportée par un module indépendant. Par conséquent, l'efficacité de ce dernier, relativement à l'alignement d'un concept présentant certaines caractéristiques particulières, peut isolément être évaluée et par la suite prise en compte lors de la combinaison de ses résultats de matching avec ceux d'autres stratégies.

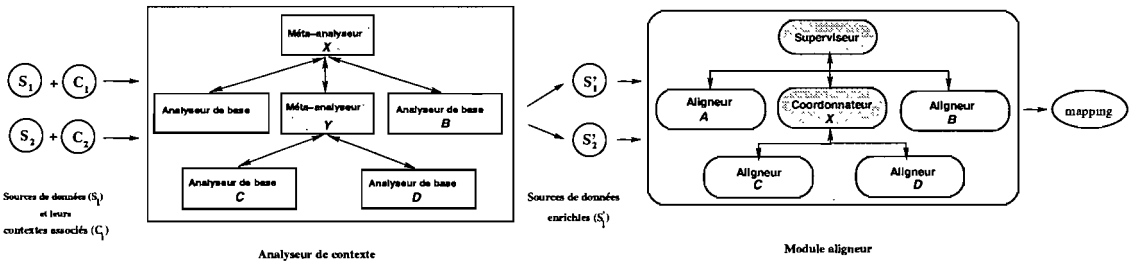


FIG. 1.1 – Processus global de l'alignement sémantique des données par le système INDIGO

Nous avons donc développé un système d'alignement des données baptisé INDIGO[2], dont l'architecture globale se compose de deux modules principaux (cf. fig. 1.2) supportant les objectifs tracés respectivement selon les deux axes de recherche. Le premier module appelé *analyseur de contexte* reçoit en entrée les sources de données à aligner accompagnées de leur contexte dont il pourra extraire des informations sémantiques destinées à enrichir ces sources de données avant de les livrer à un second module, appelé *module aligneur*, pour leur alignement proprement dit.

L'architecture de l'analyseur de contexte se compose de plusieurs modules organisés hiérarchiquement. Ce choix ayant été fait pour une analyse de contexte plus efficace et mieux maîtrisée. En effet, l'architecture comporte deux types de modules :

[2]INteroperabilty and Data InteGratiOn.

1. **Analyseurs de base** : Les analyseurs de base occupent les feuilles au sein de la hiérarchie. Ils sont spécialisés chacun dans l'analyse d'un type spécifique d'artefacts composant le contexte (e.g. programmes écrits en langage C, scripts SQL, documents textuels, etc.)
2. **Méta-analyseurs** : Les méta-analyseurs supervisent des analyseurs de base ou d'autres méta-analyseurs. Ils ont pour rôle de naviguer à travers le contexte puis de trier et aiguiller, selon leur type, les artefacts rencontrés vers chacun de leurs modules subordonnés. Chaque méta-analyseur s'occupe du tri d'une catégorie d'artefacts donnée (e.g. catégorie des programes C, C# et Java) en appliquant notamment des heuristiques (e.g. tri selon l'extension de fichier, le format interne, etc.). Il rassemble ensuite les informations sémantiques d'enrichissement retournés par ses modules subordonnés et les redirige vers le méta-analyseur qui le supervise.

Au sommet de l'architecture de l'analyseur de contexte se trouve un méta-analyseur principal qui est chargé de rassembler l'ensemble de toutes les informations sémantiques extraites à partir du contexte, en vue de procéder à l'enrichissement effectif de la source de données concernée. Deux types d'enrichissement sont à présent supportés par INDIGO. Le premier type concerne l'enrichissement lexical. Étant donné un concept issu d'une source de données (e.g. produit), l'analyseur de contexte lui associe les noms (e.g. stock, acheteur, prix, etc.) qui se retrouvent le plus fréquemment dans son voisinage au sein du contexte de la source. Quant au second type, il consiste à ajouter, à une source de données, un ensemble de combinaisons de concepts (e.g. $concat(Rue, ville, code\_postal)$, $montant * (1 + taxe)$) qui ont été identifiées en analysant son contexte (e.g. dans un programme). Ces combinaisons sont alors présentées comme de nouveaux concepts candidats pour le matching. Ces deux types d'enrichissement sont appelés à jouer un rôle de soutien de premier plan pour l'alignement sémantique des données au sein d'INDIGO. En effet, cet enrichissement lexical a pour but d'améliorer la performance du matching lexical qui établit la mise en correspondance entre les concepts en comparant notamment leurs noms respectifs. Notons que le matching lexical est largement utilisé par les systèmes actuels de mapping. En plus, il constitue une étape fondamentale dans le processus d'alignement de la majorité des solutions spécialisées dans la réconciliation des ontologies, y

compris les plus performantes d'entre elles citées dans la littérature [HCZ+06]. Or, les performances ne sont pas toujours celles attendues et nous croyons que l'enrichissement lexical permettra de corriger certaines lacunes dues au manque de précision sémantique. L'enrichissement par combinaisons de concepts, quant à lui, vise la résolution de l'un des problèmes les plus difficiles en pratique dans le domaine d'alignement sémantique, soit celui du mapping complexe. Cette difficulté est surtout liée au fait que l'espace de recherche de toutes les combinaisons possibles de concepts d'une source de données est très large, voire infini.

Une fois enrichies, les sources de données sont prises en charge par le module aligneur d'INDIGO. À l'instar de l'analyseur de contexte, le module aligneur présente aussi une architecture hiérarchique composée de deux types de modules : les aligneurs et les coordonnateurs. Chaque aligneur occupe une feuille au niveau de la hiérarchie et implémente une stratégie donnée exploitant un type spécifique d'informations sémantiques (e.g. nom de concept) pour élaborer son alignement. Un coordonnateur supervise deux ou plusieurs aligneurs et/ou coordonnateurs et se charge de la combinaison des résultats de matching renvoyés par ses modules subordonnés. INDIGO offre de ce fait plus de flexibilité, en ce qui concerne sa configuration architecturale, que les systèmes multi-stratégiques actuels puisqu'il permet entre autres la possibilité d'organiser des groupes d'aligneurs dans certains endroits choisis au niveau de la hiérarchie. Ceci peut être utile si l'on veut par exemple regrouper des aligneurs supportant des stratégies de matching différentes mais exploitant une même information sémantique sous la supervision d'un coordonnateur donné. Le module aligneur d'INDIGO offre également plus de souplesse quant à l'extensibilité car celle-ci peut se faire horizontalement en ajoutant un aligneur sous la supervision d'un coordonnateur donné, comme elle peut se faire verticalement en créant un nouveau coordonnateur avec ses aligneurs subordonnés. Enfin, il permet aussi de choisir pour chaque coordonnateur une stratégie de combinaison différente (e.g. moyenne, pondération, etc.). En particulier, le module aligneur supporte le mode de combinaison dynamique qui confère à un coordonnateur la capacité de choisir parmi ses modules subordonnés celui qui est le plus approprié à exécuter pour aligner un concept donné, considérant ses caractéristiques sémantiques spécifiques (e.g. type de données, nature de contenu, etc.). Pour cela, il s'ap-

puie sur une technique appelée *stacking* [Wol92, TW99] communément utilisée dans le domaine de l'intelligence artificielle notamment pour résoudre des problèmes de classification. Grâce à cette technique, INDIGO devient capable d'adapter sa stratégie de matching en fonction de chaque paire de sources de données à aligner. En plus, elle le rend également apte à améliorer sa performance globale au fur et à mesure qu'il cumule des expériences d'alignement (c.f. chapitre 7 pour plus de détails).

Par ailleurs, il est à noter que même si l'analyse du contexte externe des sources de données constitue une partie importante de notre système, elle n'est pas obligatoire pour l'élaboration d'un alignement. Son rôle est surtout d'améliorer la qualité de ce dernier. Ainsi, notre système présente la souplesse de traiter des sources de données qui n'offrent pas de contexte externe. Néanmoins, nous pensons qu'en pratique ces derniers sont indissociables des sources de données (e.g. le formulaire d'une application affichant les données sur écran, fait lui même déjà partie du contexte) et donc généralement disponibles.

Enfin, pour valider notre approche nous avons testé INDIGO en l'utilisant pour l'alignement de schémas de bases de données issus de quatre applications, disponibles sur Internet, de vente d'animaux en ligne. Nous l'avons également évaluée sur l'alignement de deux ontologies réelles décrivant les cours de deux universités américaines. Comme nous le verrons, les tests démontrent que l'intégration du contexte des sources de données permet effectivement d'améliorer la performance du matching lexical, et d'élaborer efficacement le mapping complexe. Par ailleurs, cette thèse montre également que le mapping adaptatif, tel qu'implémenté par INDIGO, permet de rendre notre approche multi-stratégique plus performante, voire même de surpasser trois des systèmes concurrents d'alignement parmi les plus performants cités en littérature Cupid [JPE01], Similarity Flooding SF [Mel03] et V-doc [Gro06]. L'évaluation de la performance a été basée sur la métrique f-mesure qui combine les mesures de la précision et du rappel. Étant donné un mapping de référence, la précision estime la part des matchings proposés vraiement positifs parmi l'ensemble des matchings proposés. Quant au rappel, il indique la part des matchings proposés vraiment positifs parmi tous ceux qui auraient dû être découverts.

1.3 Organisation du rapport

Le reste du présent rapport s'articule autour de huit chapitres. Le chapitre 2 aborde les *défis de l'alignement sémantique*. Nous y relatons les formes importantes d'hétérogénéité entre sources de données ainsi que les défis qu'elles soulèvent face à l'interopérabilité. On y traite aussi du Web sémantique et de quelques aspects théoriques liés au domaine de l'alignement des données. Les travaux de formalisation de la sémantique d'alignement y sont notamment discutés. Le chapitre 3 dresse une taxonomie des solutions existantes. Il met le focus sur quelques systèmes importants cités en littérature. Le chapitre 4 commence par discuter des lacunes des systèmes existants, puis expose ensuite notre approche. Les chapitres 5, 6 et 7 présentent respectivement nos trois contributions majeures de recherche dans le cadre de la présente thèse, soient la contextualisation du matching lexical, l'alignement sémantique de concepts complexes et l'adaptabilité d'une approche de mapping multi-stratégique. Après une brève présentation de l'interface utilisateur d'INDIGO, le chapitre 8 expose et analyse les résultats de nos tests expérimentaux réalisés avec INDIGO. Le dernier chapitre résume les contributions de cette thèse et conclut avec quelques perspectives de recherche.

CHAPITRE 2

LES DÉFIS DE L'ALIGNEMENT SÉMANTIQUE

L'automatisation de l'alignement sémantique présente deux obstacles majeurs : l'hétérogénéité des sources de données et le coût de la procédure d'alignement. L'hétérogénéité consistant en l'écart qui existe entre les descriptions (e.g. structures syntaxiques ou lexicales différentes) des sources à aligner rend difficile la découverte des correspondances sémantiques entre leurs concepts respectifs. Quant au coût de la procédure d'alignement, il devient important avec la taille des sources de données à aligner. En effet, plus celles-ci contiennent des nombres importants de concepts plus l'espace de recherche de tous les alignements possibles entre elles devient large, et par conséquent le coût de traitement pour la découverte des bons alignements devient aussi important. Dans la suite, avant de discuter plus en détail de ces deux obstacles, nous allons donner un aperçu des modèles de représentation des sources de données les plus couramment utilisés. Une section suivra où l'on discutera de la formalisation de l'alignement sémantique et, par la même occasion, où l'on expliquera la terminologie utilisée dans la littérature pour traiter de ces notions.

2.1 Modèles de représentation des sources de données

Comme déjà mentionné dans le chapitre 1, l'utilisation de langages standards pour la description sémantique des sources de données constitue l'une des deux principales solutions au problème d'interopérabilité sur Internet. Le Web sémantique est le projet le plus important ayant emprunté cette voie. Il est à l'origine d'une série de standards utilisés à l'échelle planétaire. La vision du Web sémantique est fondée sur le principe de rendre le contenu du Web lisible aux machines et non seulement aux humains comme c'est le cas aujourd'hui. Pour cela, il s'inspire de l'approche des systèmes de représentation de connaissances qui relèvent du domaine de l'intelligence artificielle. Ces systèmes représentent les informations d'un domaine déterminé selon une syntaxe formelle dans le but d'effectuer automatiquement des inférences sémantiques.

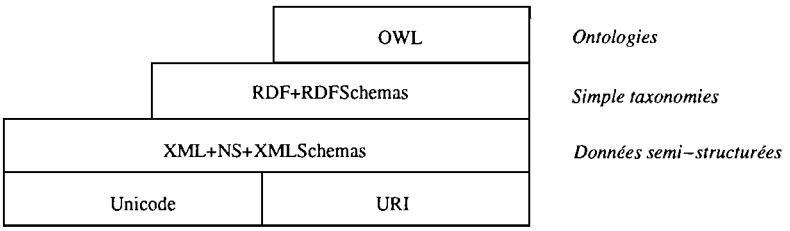


FIG. 2.1 – Les 4 premières couches du Web Sémantique

L'architecture du Web sémantique [W3C07] telle que proposée par W3C prévoit trois couches pour la *représentation de la connaissance* sur le Web. Il s'agit nommément des couches `XML+XMLSchemas`, `RDF+RDFShemas` et `Descriptions des ontologies` occupant respectivement les niveaux 2, 3 et 4 du modèle (cf. Figure 2.1).

Le Web sémantique a pour ambition de généraliser l'adoption de ces standards pour la description des données sur le Web. Il pourrait ainsi permettre l'émergence d'une panoplie d'applications intéressantes telles que :

- Recherche sémantique : la recherche de l'information basée aujourd'hui exclusivement sur le lexique des mots clés, pourra demain procéder plus intelligemment en investiguant également tous les éléments sémantiquement liés à ces mots clés.
- Négociations électroniques : dans le domaine du commerce électronique des agents seront capables de rentrer en négociation même s'ils ont des modèles de données différents.
- Intégration des données : les sources de données présentées sous forme d'ontologies sur Internet seront plus faciles à fusionner puisque la sémantique associée à leurs concepts respectifs sera plus riche et plus explicite.

Dans les trois sous-sections suivantes nous allons étudier chacun de ces standards séparément, en mettant en lumière leurs contributions respectives quant à la qualité et la nature de l'information sémantique ajoutée au contenu du Web.

2.1.1 XML

XML (eXtensible Markup Language) est un langage de balisage générique conçu pour favoriser l'échange de données sur le Web. Contrairement à HTML, il n'impose pas

de tags prédéfinis. L'utilisateur est libre de définir ses propres tags afin de définir les concepts se rapportant à son domaine particulier. Pour cela, il peut se servir de ce qu'on appelle un schéma de document XML pour définir les balises spécifiques qu'il utilisera pour décrire ses documents. Il existe deux types de schémas standards pour XML : DTD (Document Type Definition) et XSD (XML Schema Definition). Ces deux standards ont été élaborés par l'organisation W3C. Le format XSD est plus récent et donc plus évolué que DTD. En effet, XSD présente au moins deux avantages majeurs par rapport à DTD :

1. Il utilise la syntaxe XML, et peut donc bénéficier des nombreux outils de traitement du format XML (éditeur, traducteur, etc.) déjà existants. Les schémas XSD héritent également de la propriété d'extensibilité de XML (réutilisation d'un schéma dans d'autres schémas, référencement de plusieurs schémas à partir d'un seul document, etc.) ;
2. Les données étant typées, le format XSD facilite la vérification de la validité des données et la conversion entre différents types de données.

Étant donné son concept de schémas de données, le langage XML est dit auto-descriptif. Cette caractéristique est probablement à l'origine de son grand succès. En effet, elle lui confère une grande capacité d'adaptation à divers domaines.

En plus du respect des règles lexicales et syntaxiques définies par un schéma, un document XML doit être "bien formé". Autrement dit, il doit obéir à d'autres règles syntaxiques générales dont les plus importantes sont l'existence d'un seul tag englobant tout le contenu du document, la fermeture de tout tag ouvert et le non chevauchement de tags imbriqués. Un exemple de schéma XSD est donné ci-après :

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="université">
      <xs:complexType>
         <xs:element name="professeur" minOccurs="1" maxOccurs="unbounded">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="cours" type="xs:string" minOccurs="1" maxOccurs="5"/>
                  <xs:complexType>
                     <xs:attribute name="titre" type="xs:string"/>
                  </xs:complexType>
```

```
            </xs:sequence>
            <xs:attribute name="nom" type="xs:string"/>
          </xs:complexType>
        </xs:element>
      </xs:complexType>
    </xs:element>
</xs:schema>
```

Ce schéma XSD définit *université* comme élément principal de toute instance XML. Il le déclare comme type complexe pour indiquer qu'il peut contenir des éléments fils ou des attributs. Dans cet exemple, l'élément *université* peut contenir de un à plusieurs éléments *professeur* tel qu'indiqué par les valeurs affectées aux mots clés *minOccurs* et *maxOccurs* associés à cet élément. L'élément *professeur* est lui-même de type complexe et peut contenir de 1 à 5 éléments *cours* de même qu'un attribut de type *string* appelé *nom*. Enfin, chaque élément *cours* possède un attribut de type string appelé *titre*.

Le document XML ci-après se présente comme une instance du schéma XSD ci-dessus. Il définit un *professeur* appelé `André Gagnon` qui enseigne les cours d'histoire et de géographie.

```
<?xml version="1.0"?>
    <université xmlns="http://www.w3schools.com"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.w3schools.com univ.xsd">
          <professeur nom="André Gagnon">
               <cours titre="Histoire"/>
               <cours titre=" Géographie "/>
          </professeur>
    </université>
```

Comme nous pouvons le constater, XML permet de structurer les données par l'utilisation de tags dont l'interprétation n'est pas formellement définie mais est plutôt celle sous-entendue par leur concepteur. A ce titre, il est bien adapté pour jouer le rôle de support d'échange de données. Un langage tel HTML est moins approprié pour jouer ce rôle puisqu'il n'est pas générique. En effet, il véhicule des informations spécifiques concernant la présentation visuelle des données dans un document.

Par ailleurs, des outils basés sur XSLT existent pour transformer les documents XML d'un schéma vers un autre. Entre autres, grâce à XSLT, un même document XML

peut être systématiquement converti en plusieurs autres formats tels que XHTML (page Web), SMIL (Synchronized Multimedia Integration Language), etc. Ceci constitue donc un moyen pour séparer les données de leur présentation.

En revanche, malgré toutes ses vertus, XML reste Pauvre du côté sémantique. Ceci est en fait un revers de sa capacité "auto-descriptive". En effet, ne prédéfinissant pas de tags ayant une sémantique standard, il ne permet pas de réaliser une description sémantique de données qui peut être interprétée de façon non ambigüe à l'échelle universelle.

De plus, XML ne propose pas d'approche particulière pour la description des données. En revenant à l'exemple décrivant la relation entre les professeurs d'une université et des cours qu'ils donnent, il existe plusieurs façons de l'exprimer avec un schéma XSD. En effet, il est possible de proposer une représentation centralisée sur les cours plutôt que sur les professeurs. On obtiendra alors une version différente du schéma précédent :

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="université">
      <xs:complexType>
         <xs:element name="cours" minOccurs="1" maxOccurs="unbounded">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="professeur" type="xs:string"/>
                  <xs:complexType>
                     <xs:attribute name="nom" type="xs:string"/>
                  </xs:complexType>
               </xs:sequence>
               <xs:attribute name="titre" type="xs:string"/>
            </xs:complexType>
         </xs:element>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

Cette souplesse au niveau de la représentation complexifie l'interprétation sémantique des sources de données décrites en XML. Prenons comme exemple les deux sources de données schématisées par la figure 2.2 : elles décrivent des concepts similaires en

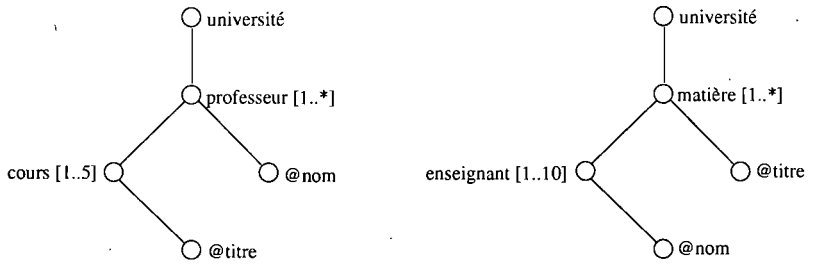


FIG. 2.2 – Exemple de deux documents XML différents modélisant pourtant les mêmes concepts

utilisant des lexiques différents, (e.g. *enseignant* correspondant à *professeur* et *matière* à *cours*). En outre, même si elles contiennent des concepts équivalents, elles les présentent selon des modèles structuraux différents : tandis que l'un liste les professeurs à un niveau hiérarchique supérieur que celui des cours correspondants, l'autre liste les enseignants comme des sous-éléments des matières associées.

Ainsi, si l'on souhaite par exemple éditer les noms de tous les professeurs enseignant l'histoire, la requête XQuery à formuler sur une source de données sera différente de celle à appliquer sur l'autre source : elles seront respectivement `/université/professeur [cour/@titre= "histoire"]/@nom` et `/université/matière[@titre= "histoire"] /enseignant/@nom`.

En conclusion, XML est un langage adapté pour la structuration des données sur le Web, notamment à des fins d'échange, mais ne convient pas à l'expression de leur sémantique. Pour cette raison, le modèle du Web sémantique prévoit une autre couche nommée `RDF+RDFShemas` au dessus de `XML+XMLShemas` destinée spécifiquement à la description sémantique des données.

2.1.2 RDF

`RDF+RDFShemas` est la couche qui se situe immédiatement au dessus de `XML+XMLShemas`. Cette couche propose un modèle doté d'une sémantique simple pour la représentation des ressources sur internet. La notation RDF repose sur les choix stratégiques suivants :

1. L'utilisation des URIs (Uniform Resource Identifiers) pour permettre une iden-

tification unique des ressources sur le Web. Une ressource peut être n'importe quelle entité accessible via le Web, que ce soit une simple donnée élémentaire ou un site Web. Un URI se compose de trois parties : (1) le nom du protocole Internet utilisé pour accéder à la ressource, (2) l'adresse de la machine où se trouve la ressource et (3) le chemin spécifique menant directement à cette ressource. Par exemple, les trois parties composant l'URI de la page Web personnelle du professeur André Gagnon seront respectivement `http ://` (Hypertext Transfer Protocol), `www.iro.umontreal.ca` et `/Gagnon`.

2. L'utilisation d'un modèle très simple et intuitif pour la description des ressources. Ce modèle met en relation trois types d'entités appelées *sujet*, *prédicat* et *objet* (souvent désigné aussi par *ressource*, *propriété* et *valeur*). Cette organisation en triplet s'apparente à la façon dont un humain décrit naturellement les choses (i.e. sujet, verbe et complément). Par exemple, l'affirmation "la page Web `http ://www.iro.umontreal.ca/ aGagnon` a été créée par `André Gagnon`" sera exprimée en RDF par l'utilisation du triplet `http ://www.iro.umontreal.ca/Gagnon` (ressource), `http ://www.schema.org/#Creator` (propriété) et `André Gagnon` (valeur). Les expressions RDF sont représentées par un graphe dirigé (cf. figure 2.3). En fait, le méta-modèle de RDF peut être décrit par un schéma XSD. Il est donc naturel de décrire les modèles RDF en XML, bien qu'une autre notation aurait pu être choisie.

```
<?xml:namespacens="http://www.w3.org/TR/WD-rdf-syntax" prefix="RDF"?>
<?xml:namespacens="http://purl.org/metadata/dublin_core" prefix="DC"?>
      <RDF:RDF>
            <RDF:Descriptionabout=" http://www.iro.umontreal.ca/~aGagnon ">
                  <DC:Creator>André Gagnon</DC:Creator>
            </RDF:Description>
      </RDF:RDF>
```

RDF permet d'établir un lien logique entre ressources, propriétés et valeurs présentes sur le Web. Chaque ressource est clairement identifiée par un URI unique. Les propriétés peuvent aussi être identifiés à l'aide des URIs et jouer à leur tour le rôle de ressources dans d'autres expressions RDF. Il devient ainsi possible de réaliser des tissages logiques entre concepts, propriétés ou toute autre ressource existante sur le Web.

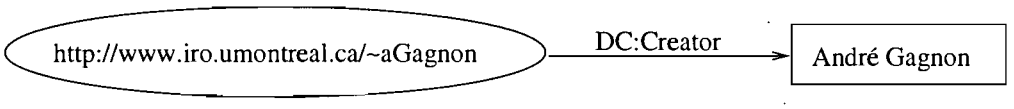


FIG. 2.3 – Exemple de modèle RDF

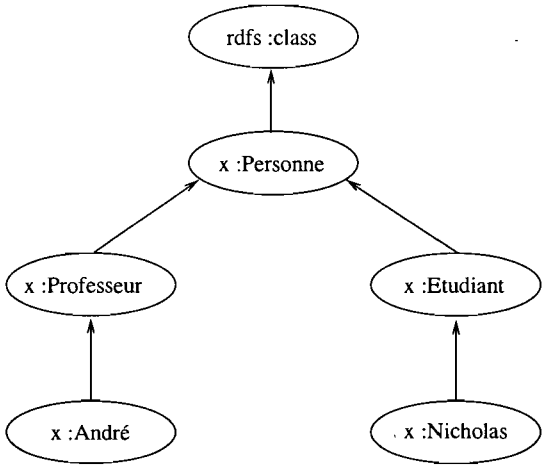


FIG. 2.4 – Exemple de classes d'un Schéma RDF

A l'instar des modèles XML qui sont décrits par des schémas XSD, un modèle RDF est décrit par un schéma RDFS. RDFS permet de définir des classes et des relations de généralisation entre classes. La propriété rdfs :class est utilisée pour décrire les classes alors que la propriété rdfs :subClassOf est utilisée pour le sous-classement (cf. Figure 2.4). Dans RDF, la propriété `rdf :type` est utilisée pour indiquer qu'une ressource est une instance d'une classe.

Par ailleurs, la propriété `rdfs :subPropertyOf` est utilisé pour indiquer qu'une propriété est une sous-propriété d'une autre (cf. Figure 2.5) ; ce qui implique que toute paire de ressources reliée par la première propriété le sont aussi par la seconde. RDFS permet également de définir des contraintes sur les propriétés. Par exemple, `P rdfs :domain C` indique que toute ressource possédant la propriété `P` est une instance de la classe `C` ; et `P rdfs :range C` indique que les valeurs de la propriété `P` sont des instances de la classe `C`.

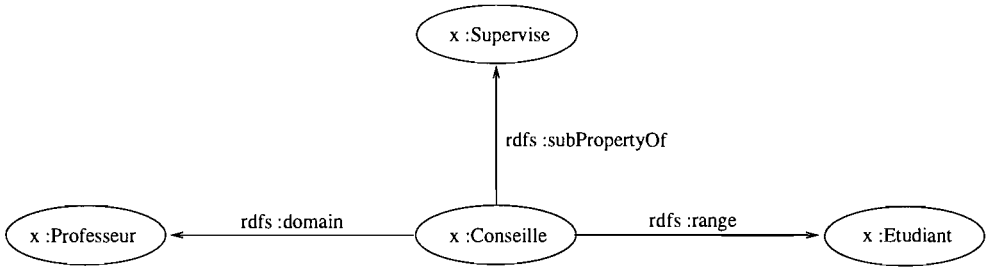


FIG. 2.5 – Exemple de propriétés d'un Schéma RDF

Grâce à son mécanisme de triplets, RDF offre une plateforme adéquate pour l'inférence. Par exemple, des deux affirmations "André Gagnon habite au code postal H4T 7F6" et "le code postal H4T 7F6 se trouve au Canada" un programme d'inférence basé sur RDF permet de déduire que "André Gagnon habite au Canada". En effet, les triplets RDF (André Gagnon, code postal, H4T 7F6) et (H4T 7F6, pays, Canada) traduisant respectivement les deux affirmations initiales montrent que H4T 7F6 joue le rôle de valeur de propriété dans le premier et de ressource dans le second. Il représente donc un point de jonction des deux graphes RDF modélisant ces deux triplets. L'utilisation d'un langage tel que SPARQL (Simple Protocol And RDF Query Language) basé sur un interpréteur de triplets permet entre autres, de générer un sous-graphe modélisant le triplet (André Gagnon, Résident, Canada). En effet, SPARQL est un langage de requête de base de données de type RDF. Il est actuellement en cours de normalisation par l'organisation W3C. SPARQL est adapté à la structure spécifique des graphes RDF et supporte les deux types de requêtes suivants :

- Les requêtes interrogatives basées sur la clause SELECT, permettent d'extraire à partir d'un graphe RDF un sous-graphe composé de ressources répondant à un ensemble de critères définis dans la clause WHERE.
- Les requêtes contructives basées sur la clause CONSTRUCT, permettent de générer un nouveau sous-graphe d'un graphe RDF.

L'exécution de la requête constructive ci-après, permet de générer le sous-graphe modélisant le triplet (André Gagnon, résident, Canada) à partir du graphe RDF modélisant les deux triplets (André Gagnon, code postal, H4T 7F6) et (H4T 7F6, pays, Canada).

```
CONSTRUCT ?v1    c:résident      ?v3

WHERE {
        ?v1    c:code_postal   ?v2
        ?v2    c:pays          ?v3
      }
```

En résumé, RDF permet de décrire les méta-données et chaque schéma RDFS introduit un modèle de données particulier avec un vocabulaire précis pour décrire ces données selon la structure RDF. Ainsi, par rapport à XML qui est très simple mais aussi conceptuellement très pauvre, l'association RDF/RDFS permet de franchir un pas important vers l'intégration de la sémantique aux sources de données. Elle permet par exemple de déclarer que :

- `Personne`, `Professeur` et `Cours` sont des classes,
- `Professeur` est une sous classe de `Personne`,
- `Enseigner` est une propriété reliant les classes `Professeur` et `Cours`,
- `Age` est une propriété ; avec `Personne` est son domaine et `Entier` son type,
- et, *André Gagnon* est une instance de `Professeur` et que son age est 36 ans.

Toutefois, `RDF/RDFS` présente encore des limitations par rapport à son expressivité sémantique. A titre d'exemple, il ne permet pas d'exprimer le fait qu'une classe est disjointe d'une autre ou encore qu'une propriété est l'inverse d'une autre. Le modèle du Web sémantique propose une troisième couche dénommée `Description des ontologies` qui se situe au dessus de la couche `RDF/RDFS`, pour apporter un meilleur support à la description sémantique des données.

2.1.3 OWL

Dans le contexte du Web sémantique, l'objectif principal des ontologies est de présenter les données sur Internet sous une forme qui soit adaptée pour le raisonnement

logique de programmes, en l'occurrence par des agents intelligents.

Typiquement, une ontologie se définit comme une taxonomie de concepts associée à un ensemble de règles d'inférence. Pour cette raison, le point commun entre les différents langages ontologiques est leur utilisation des logiques de descriptions.

OWL est un nouveau langage développé par W3C, dans la même lignée que XML et RDF, pour la prise en charge des descriptions ontologiques dans le cadre du Web sémantique. Il est appelé, entre autres, à faciliter la communication entre agents en établissant un vocabulaire commun entre eux. Il vise également à améliorer la performance de la recherche d'informations sur Internet en permettant d'identifier les concepts liés sémantiquement aux mots clés de recherche, et non seulement ceux liés lexicalement.

Par ailleurs, la conception de OWL tient compte des expériences précédentes dans le domaine de définition des ontologies. Elle a été principalement influencée par les modèles de OIL, DAML+OIL et RDF. En particulier, étant construit sur la couche RDF/RDFS du modèle du Web Sémantique, OWL se présente comme une extension des services offerts au niveau de cette couche. Ainsi, en plus des mécanismes comme la déclaration de classes et propriétés ou instanciation, subsomption, définition des domaines et domaines de validité, OWL offre les fonctionnalités suivantes :

1. La spécification d'une classe comme une combinaison d'intersection, réunion ou complément d'autres classes.
2. La spécification d'une classe comme une énumération d'objets spécifiques d'autres classes.
3. La définition des domaines de validité des propriétés soit comme des classes OWL ou bien des types externes de données comme `string` ou `integer`.
4. La déclaration d'une propriété comme transitive, symétrique, fonctionnelle, ou comme l'inverse d'une autre propriété.
5. La possibilité de spécifier quels individus (objets) appartiennent à quelles classes et quelles sont les valeurs de propriétés liées à des individus spécifiques.
6. La possibilité d'exprimer que deux classes ou bien deux propriétés sont équivalentes.
7. La possibilité d'exprimer que deux classes sont disjointes.

8. La possibilité d'affirmer l'égalité ou l'inégalité entre deux individus.
9. La possibilité de définir des contraintes sur les valeurs d'une propriété particulière d'une classe telles que :

   - Toutes les valeurs de la propriété dans les instances de la classe doivent appartenir à une classe ou à un type de données.

   - Au moins une valeur doit provenir d'une certaine classe ou d'un type de données.

   - Il doit y avoir un minimum ou un maximum de valeurs distinctes.

Avec OWL, il est possible de spécifier par exemple que :

- Les classes `Personne` et `Université` sont disjointes.
- *André* et *Nicolas* sont des individus distincts.
- `Encadrer` est l'inverse de la propriété est `supervisé par`.

2.2 Définition formelle de l'alignement sémantique

Une définition formelle du mapping sémantique est essentielle pour un développement plus maîtrisé des solutions aux problèmes d'alignement de données. En effet, cette définition permettra entre autres de répondre aux besoins suivants :

1. Expliciter et par conséquent mieux cerner toutes les notions importantes caractérisant une application d'alignement des données : types d'informations utilisés en entrée (e.g. schémas , instances, etc.), formules de calcul de similarité utilisées, types de mapping fournis en sortie, etc.
2. Disposer d'une notation formelle pour la description des mappings : ceci pemettra de comprendre de façon précise et non ambigüe les différentes approches utilisées par les applications d'alignement des données, de les comparer et de les améliorer en conséquence.

Toutefois, la majorité des systèmes d'alignement des données présents dans la littérature ne présentent pas la définition du type de mapping qu'ils implémentent. Leurs auteurs ont plutôt mis l'accent sur le test et les résultats des démarches proposées. Néanmoins,

il existe plusieurs travaux qui se sont intéressés à la définition formelle du problème [BC86, Len02, MHDB02, Doa02]. Dans ce qui suit, nous allons expliquer la notion de l'alignement sémantique au moyen d'un exemple avant d'aborder la formalisation de sa définition.

2.2.1 Exemple d'un alignement sémantique

Rappelons que l'alignement sémantique entre deux sources de données consiste en l'ensemble des mises en correspondance sémantiques qui existent entre leurs concepts respectifs. La figure 2.6 illustre un exemple d'alignement sémantique entre deux tables d'employés nommées repectivement *Employes* et *Personnel*. Prenons par exemple le concept nommé *Matricule* de la table *Employes*, il a été mis en correspondance avec le concept nommé *Numéro* de la table *Personnel*. En effet, ces concepts représentent tous les deux le code attribué à chaque employé pour l'identifier de façon unique au sein de l'entreprise. Il est à noter que ce type d'alignement qui est de cardinalité *un à un* est communément appelé *matching simple*. Par contre, l'alignement mettant en correspondance le concept *Adresse* de la table *Employes* avec le triplet *(Rue, Ville, Code_Postal)* de la table *Personnel* est appelé *matching complexe*. Il est à noter toutefois que dans la pratique ce triplet est généralement exprimé comme une combinaison de termes auxquels sont appliqués un ou plusieurs opérateurs. Pour le cas particulier du matching du concept *Adresse*, l'opérateur utilisé est ici celui de la concaténation de chaînes de caractères appliquée aux concepts *Rue*, *Ville* et *Code_Postal*. Nous aurons donc le matching complexe suivant *Adresse* $\rightarrow$ *concat(Rue, Ville, Code_Postal)*. Il est à noter que dans sa forme la plus générale, le *matching complexe* ne se limite pas aux mappings de 1 à n. Il concerne la mise en correspondance entre une combinaison de concepts sources et une combinaison de concepts cibles et par conséquent peut donner lieu aux trois cardinalités suivantes *1 à n*, *n à 1* et *n à m*. Jusqu'à présent, peu de travaux se sont intéressés au maching complexe. Car, en pratique, il est plus difficile à élaborer que le matching simple. En effet, l'espace de recherche des combinaisons source et cible peut être très large, voire théoriquement infini [RLD+04].

Par ailleurs, notons que dans la littérature, les termes *matching* et *mapping* sont

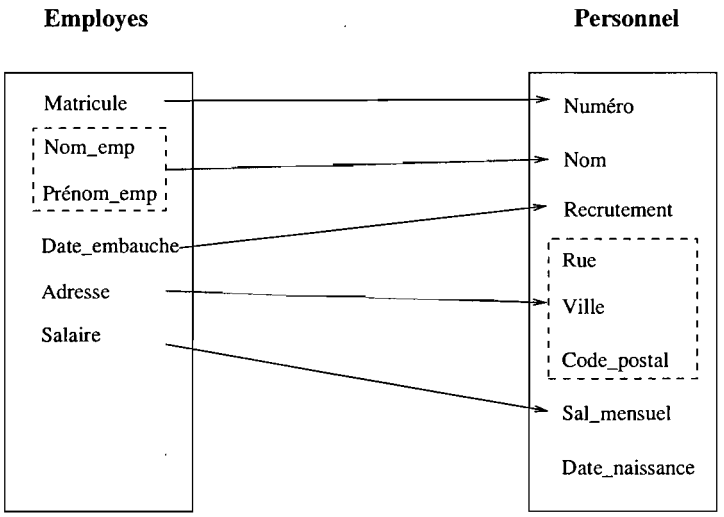


FIG. 2.6 – Exemple d'alignement sémantique

souvent employées indifféremment pour désigner l'alignement sémantique entre concepts de sources de données. Il arrive toutefois qu'on leur attribue des sens différents :

- le *matching* désigne toujours la mise en correspondance entre concepts sources et concepts cibles partageant une même sémantique,
- et, le *mapping* définit parfois la formule de transformation effective de la source vers la cible en s'appuyant sur un matching. La formule peut être exprimée par exemple sous forme d'une requête SQL ou plus généralement comme une expression de logique de premier ordre.

Dans la suite du rapport, nous allons plutôt nous conformer à la convention initiale considérant que le sens porté par le *mapping* coïncide avec celui du *matching*.

2.2.2 Formalisation de la notion du mapping en littérature

De façon générale, nous pouvons distinguer dans la littérature deux catégories de travaux portant sur la formalisation du mapping. La première regroupe ceux qui ciblent un domaine d'application bien défini. Le cas typique concerne les définitions ciblant les applications de bases de données et plus précisément les systèmes d'intégration de données. La seconde catégorie concerne les travaux visant la définition d'un cadre de

formalisation générique qui peut convenir à plusieurs types d'applications d'alignement des données.

2.2.2.1 Définition formelle du mapping dans un contexte spécifique

[BC86] et [Len02] comptent parmi les travaux à noter au sein de cette catégorie. Ils s'intéressent au problème de formalisation dans le cas d'applications d'intégration de bases de données. Ce type d'application consiste à intégrer au sein d'une seule base de données globale, un ensemble de sources de données locales. En pratique, on envisage d'intégrer les bases de données par la simple construction d'une vue globale plutôt que de consolider physiquement l'ensemble des données locales dans une même base. Dans ce cas, on parle plutôt d'une intégration de vues locales en une vue globale. L'intégration devra en particulier, tenir compte des contraintes d'intégrité définies localement. L'objectif est de permettre aux utilisateurs d'obtenir, de façon transparente, les mêmes résultats en interrogeant la vue globale que s'ils avaient interrogé les sources locales et ce, en utilisant les mêmes requêtes.

Dans [Len02], l'auteur aborde le problème d'intégration des données dans un cadre plus général. Il se base aussi sur l'idée de combiner des données présentes dans différentes sources locales pour élaborer une vue unifiée globale. Les sources contiennent des données réelles tandis que le schéma global se présente sous forme d'une vue virtuelle. L'auteur s'intéresse particulièrement aux aspects de modélisation des applications d'intégration des données, au traitement des requêtes dans le contexte d'intégration des données, à la gestion de l'inconsistence entre sources de données et au raisonnement sur les requêtes.

La modélisation concerne le mapping entre les sources de données et le schéma global. Le mapping servira de base pour la traduction d'une requête globale en des requêtes locales. La modélisation s'appuie sur deux approches fondamentales souvent décrites dans la littérature. La première, appelée *global-as-view* (**GAV**), consiste à définir le schéma global à partir des sources locales. La seconde approche, appelée *local-as-view* (**LAV**), consiste à spécifier le schéma global indépendamment des sources locales dans un premier temps. Ce n'est qu'ensuite que les relations entre le schéma global et les

sources locales sont établies en définissant chaque source comme une vue sur le schéma global. En somme, les composantes principales d'un système d'intégration des données sont le schéma global, les sources locales et le mapping.

Ainsi, un système d'intégration des données I est formalisé par [Len02] sous forme d'un triplet $< G, S, M >$ tel que :

- G est le schéma global, exprimé dans un langage L_G basé sur un vovabulaire V_G. V_G définit un symbole pour chaque élément de G (e.g. *relation* si G est un modèle relationnel, classe si G est modèle orienté objet, etc.).
- S est le schéma source, exprimé dans un langage L_S basé sur un vovabulaire V_S. V_S prévoit un symbole pour chaque élément de S.
- M est le mapping entre G et S, composé par un ensemble d'assertions de la forme :

$$q_g \rightarrow q_s$$
$$q_s \rightarrow q_g$$

où q_s et q_g sont deux requêtes avec la même arité, appliquées respectivement sur le schéma source S et le schéma global G. Les requêtes q_s sont exprimées dans un langage de requêtes $L_{M,S}$ basé sur le vocabulaire V_S, et les requêtes q_g sont exprimées dans un langage de requêtes $L_{M,G}$ basé sur le vocabulaire V_G. Intuitivement, une assertion $q_s \rightarrow q_g$ spécifie que le concept représenté par la requête q_s sur les sources correspond au concept dans le schéma global représenté par la requête q_g (idem pour $q_g \rightarrow q_s$).

Dans un système d'intégration des données $I =< G, S, M >$ basé sur une approche **LAV**, le mapping M associe à chaque élément s du schéma source S une requête q_g sur le schéma global G. Autrement dit, le langage des requêtes $L_{M,S}$ autorise seulement les expressions constituée par des symboles du vocabulaire V_S. Ainsi, un mapping **LAV** est un ensemble d'assertions, une pour chaque élément s de S, de la forme $s \rightarrow q_g$. D'un point de vue modélisation, l'approche **LAV** est basée sur l'idée que le contenu de chaque élément source s doit être défini en terme d'une vue q_g sur G. L'approche **LAV** est la plus appropriée lorsque le système d'intégration des données est basé sur un

schéma global stable. Il est à noter qu'elle facilite l'extensibilité du système : l'ajout d'une nouvelle source requiert tout simplement l'enrichissement du mapping par de nouvelles assertions.

Dans une approche **GAV**, le mapping M associe à chaque élément g dans G une requête q_s définie sur une source S. En d'autres termes, le langage de requête $L_{M,G}$ autorise seulement les expressions composées par des symboles de V_G. Ainsi, un mapping **GAV** est un ensemble d'assertions, une pour chaque élément g du schéma G, de la forme $g \rightarrow q_s$. D'un point de vue modélisation, l'approche **GAV** est basée sur l'idée que le contenu de chaque élément global g doit être défini en terme d'une vue q_s.

Il est généralement reconnu que le traitement des requêtes dans le cas d'une approche **LAV** est une tâche complexe. En effet, dans cette approche, la seule connaissance qu'on a sur les données du schéma global réside dans les vues sur les sources locales ; et de telles vues ne fournissent que de l'information partielle sur les données. Puisque le mapping associe à chaque source une vue sur le schéma global, il n'est pas évident de déterminer comment utiliser les sources pour répondre à une requête exprimée sur le schéma global. À l'inverse, le traitement des requêtes semble plus facile dans le cas d'une approche **GAV**. Car, le mapping spécifie directement quelle requête source utilisée pour trouver un élément du schéma global.

2.2.2.2 Approches génériques

Parmi les travaux supportant les approches génériques, notons ceux de [MHDB02] et [Doa02]. Leur objectif est la définition d'un cadre générique pour la spécification de langages de mapping et de la sémantique qui leur est associée. Dans [MHDB02], le cadre de formalisation proposé par les auteurs intègre trois caractéristiques importantes :

- Il supporte un mapping direct entre des modèles définis dans des langages différents (e.g. XML, RDF, etc.).
- Il utilise *un modèle auxiliaire* dans le cas où il est impossible d'établir un mapping direct entre deux modèles.
- Il permet de représenter des mappings incomplets ou qui manquent d'information.

Les auteurs définissent le mapping entre modèles comme un ensemble de comparaisons entre deux ensembles d'expressions. Les expressions sont écrites dans un langage prédéfini, basé sur un ensemble d'opérateurs. Pour que les deux expressions soient comparables, il faut que les types des résultats qu'elles renvoient soient de même nature (e.g. égalité de deux nombres, inclusion entre deux ensembles, etc.). Par exemple, si une expression se présente sous forme d'une requête SQL alors elle renverra une relation (ensemble d'enregistrements) ; par contre si elle est définie comme une requête XQuery, elle produira une structure arborescente (document XML). Le langage de définition des expressions utilisées dans un mapping varie en fonction des langages de spécification des modèles à aligner.

D'un point de vue formel, étant donné deux modèles : M_1 (décrit dans un langage L_1) et M_2 (décrit dans un langage L_2), une formule définie sur M_1 et M_2 sera de la forme e_1 **op** e_2, où e_1 et e_2 sont des expressions, et **op** est un opérateur défini en fonction des types des résultats retournés par e_1 et e_2. Par exemple, l'opérateur **op** peut être $=$ ou $\subseteq$ pour mettre en relation des expressions s'évaluant par tables relationnelles, comme il peut être $=$, $\geq$ ou $\leq$ si les résultats sont des nombres, etc.

Par ailleurs, un mapping entre deux modèles M_1 et M_2 peut aussi inclure un modèle auxiliaire M_3 (décrit dans un langage L_3). Dans ce cas, il sera défini comme un ensemble de relations sur *(M_1,M_2)*, *(M_1,M_3)* et *(M_2,M_3)*. L'utilisation d'un modèle auxiliaire est justifié lorsqu'il existe, par exemple, une correspondance sémantique entre deux concepts *C1* et *C2* (e.g. désignés par des noms similaires), mais il existe des contraintes d'intégrité imposant que l'ensemble des valeurs de *C1* soit disjoint de celui de *C2* (e.g. C.I Source : S.ville='Montréal' et C.I Cible : C.ville='Québec'). Dans ce cas, *C1* et *C2* seront tous les deux alignés à un même *C3* défini au sein d'un modèle auxiliaire.

Enfin, il est à noter que les auteurs de [MHDB02] ont aussi défini un ensemble de propriétés permettant de vérifier la validité d'un mapping dans le contexte d'une application réelle :

Consistance : un mapping est consistant lorsqu'il permet d'obtenir les résultats attendus suite à une requête donnée. Par exemple, dans le cas d'une application d'intégration des données, les requêtes appliquées au schéma central renvoient les

mêmes résultats que si elles étaient appliquées directement aux sources de données locales.

Optimalité : Un alignement entre deux sources de données est optimal s'il est complet et minimal. Un mapping complet est celui qui inclut toutes les mises en correspondance définies dans un mapping de référence. Il est considéré minimal si l'ensemble de ses mises en correspondances est inclus dans celui du mapping de référence. Par conséquent, un mapping optimal correspond au mapping de référence.

Composabilité : la composabilité est la capacité de générer un mapping par composition d'autres mappings. Le cas typique concerne la vérification de la transitivité : étant donné trois modèles de données M_1, M_2 et M_3, il est important de vérifier si à partir des deux mappings (M_1,M_2) et (M_2,M_3) il serait possible de composer le mapping (M_1,M_3).

2.3 Hétérogénéité des sources de données

Comme déjà discuté par la section 2.1, il existe actuellement plusieurs modèles pour présenter les données sur le Web. Chaque modèle se caractérise par sa syntaxe et son vocabulaire spécifiques. Ceci contribue ainsi à l'hétérogénéité des sources de données présentes sur l'Internet, pour les raisons suivantes :

1. **Incompatibilité structurelle :** les données peuvent être structurées différemment d'un modèle à un autre. Considérons par exemple le modèle relationnel et le format XML. Le premier modèle représente les concepts par des attributs qu'il regroupe au sein d'entités inter-reliées par la notion de «clé étrangère». Tandis que le second utilise le mécanisme d'imbrication pour exprimer des relations entre ces entités.
2. **Écart expressif :** certaines notions sémantiques peuvent être supportées par certains modèles et non pas par d'autres. Par exemple, la notion de classe qui peut être définie par OWL n'est pas supportée par XML.

Toutefois, il existe des facteurs autres que ceux liés à la diversité des modèles, causant aussi l'hétérogénéité des sources de données. Nous en citons ci-après les cas les plus fréquents en pratique :

1. **Synonymie :** deux concepts ayant des noms différents peuvent pourtant partager une même sémantique. Par exemple, les concepts nommés *téléphone_bureau* et *téléphone_travail* désignent tous les deux, le numéro de téléphone au travail d'un employé même si leurs nom respectifs sont lexicalement différents.
2. **Polysémie :** la polysémie est la notion inverse de la synonymie. Elle correspond au cas où deux concepts qui possèdent un même nom mais portent des sémantiques différentes. Par exemple, deux concepts appelés tous les deux *téléphone* mais désignant respectivement le téléphone au domicile et le téléphone au travail.
3. **Abréviation :** il n'est pas rare de trouver des noms abrégés de concepts dans les sources de données de la vie réelle. Par exemple, le concept «adresse d'un employé» s'appellerait *adr* et le concept «Téléphone» se nommerait *tel.*

Enfin, l'hétérogénéité entre deux sources de données peut aussi provenir simplement du fait que celles-ci ne décrivent pas les mêmes ensembles de concepts. Par exemple, une source de données peut décrire plusieurs types de commandes (demande, confirmation, etc.) alors qu'une autre n'en déclare qu'un seul. Un autre exemple à citer concerne la librairie xCBL (XML Common Business Library) qui a été créée par `Commerce One` en association avec des chefs de file dans le domaine du commerce électronique B2B. Cette librairie définit un ensemble de schémas XML de documents d'affaires appelés à être utilisés par des acteurs opérant sur une place de marché électronique. Afin de supporter la diversité des besoins de ces acteurs, les schémas xCBL proposés ont pris des tailles importantes et des structures très complexes. En particulier, il y a une trentaine d'éléments définis juste dans l'entête de l'objet commande *Order.* Un écart important s'est donc créé entre la librairie standard xCBL et les schémas de données des différents acteurs. Ceci rend bien entendu difficile l'alignement entre ces schémas locaux et le schéma central de xCBL.

2.4 Le coût de la procédure

Comme déjà mentionné dans le chapitre 1, le coût du matching simple nécessite $n*m$ comparaisons si les nombres de concepts des sources de données de départ et d'arrivée

sont respectivement n et m. En effet, un alignement de cardinalité *un à un* consiste à comparer chaque concept de la source de données d'arrivée avec tous les concepts de celle de départ. Après comparaison, les paires de concepts doivent être triés par ordre décroissant de similarités. Seules les paires dont les valeurs de similarités associées dépassent un certain seuil prédéfini seront retenues. Si l'on suppose l'utilisation d'un algorithme de tri aussi performant que *QuickSort* pour la classification des paires de concepts, le coût du traitement global sera de l'ordre de *n*m*log(n*m)*. Encore faut-il que le coût atomique d'une comparaison soit majoré par une constante (e.g. comparaison des noms de concepts basée sur métrique de similarité lexicale). En effet, si la comparaison des paires de concepts implique un algorithme plus complexe (e.g. comparaison de paramètres statistiques de contenus de champs de bases de données) le coût de la procédure pourra être encore plus élevé dépendamment du degré de complexité de l'algorithme adopté et du volume des données traitées.

Par ailleurs, le matching complexe soulève un problème de coût de traitement beaucoup plus important. En fait, tandis que l'espace de recherche des candidats pour le matching simple est fini, il peut être très large dans le cas d'un matching complexe. En effet, on peut imaginer un nombre important d'opérateurs *(concat, +, *, ...)* qui peuvent être utilisés pour combiner les concepts d'une source de données, ce qui donne lieu à de nombreuses combinaisons possibles.

2.5 Conclusion

L'alignement sémantique des données lance deux défis importants : pallier l'hétérogénéité des sources de données et maîtriser le coût de la procédure de matching. L'hétérogénéité des sources de données est essentiellement due à la diversité des modèles adoptés pour présenter celles-ci sur le Web et à celle des formes lexicales utilisées pour désigner les concepts qui les composent. Quant au coût de la procédure d'alignement, il est surtout lié à la taille des sources de données à aligner et pose un problème important dans le cas particulier du matching complexe. Le chapitre suivant fait l'état de l'art des solutions ayant relevé ces deux défis et tenté d'automatiser le processus d'alignement sémantique.

CHAPITRE 3

ÉTAT DE L'ART

Il existe aujourd'hui plusieurs travaux de recherche [Noy04, EBB+04] qui se sont intéressés au problème d'automatisation de l'alignement sémantique des données. Ces travaux ont abouti au développement d'une panoplie de systèmes qui diffèrent par leurs architectures, leurs approches d'alignement et les types de modèles de données supportés. Ce chapitre présente un échantillon représentatif de 6 systèmes faisant partie de ceux les plus cités dans la littérature. Il en propose ensuite une classification basée sur cinq critères inspirés des meilleures pratiques dans le domaine d'alignement sémantique des données.

3.1 LSD

LSD [DDH03] est un système conçu pour aligner un ensemble de sources de données locales avec un schéma central. Il est spécialisé dans l'appariement des modèles XML. Pour élaborer un mapping, **LSD** exploite les instances de valeurs associées aux concepts composant les sources de données et les informations liées à leurs schémas respectifs. Pour cela, il a exclusivement recours aux techniques d'apprentissage. En fait, il emploie plusieurs modules d'alignement indépendants appelés apprenants de base. Chaque apprenant de base est spécialisé dans le traitement d'une ressource d'information sémantique spécifique. L'architecture du système **LSD** telle que présentée dans [DDH03] comprend quatre apprenants de base dont deux exploitant les contenus de concepts, un traitant leurs noms et enfin un dernier spécialisé dans l'analyse des structures des sources de données à aligner. Les apprenants de base sont supervisés par une unité spéciale appelée méta-apprenant qui combine leurs résultats respectifs pour générer le mapping final. Pour cela, le méta-apprenant utilise une technique appelée *stacking* [Wol92, TW99] qui permet d'évaluer le poids à attribuer à chaque apprenant de base par rapport à un concept cible donné. Ainsi, le processus d'alignement du système **LSD** s'exécute en deux

phases principales :

Phase d'entraînement. Chaque apprenant de base est entraîné sur un ensemble de données alignées manuellement pour générer son propre modèle de classification. Pour cela, un expert utilisateur effectue un mapping manuel entre un ensemble de sources de données choisies arbitrairement et le schéma central. Ensuite, pour chaque apprenant de base, des données d'entraînement spécifiques sont extraites de ce mapping. Par exemple, un apprenant spécialisé dans le traitement des noms de concepts recevra des données sous la forme d'un ensemble de paires *(nom de concept source, nom de concept cible)*, tandis qu'un apprenant de contenus se verra fournir des paires *(instance source, nom de concept cible)*. Pour le cas particulier d'un apprenant de noms, les noms de concepts sources sont fournis sous forme de chemins XML complets intégrant tous les tags allant de la racine jusqu'au noeud représentant le concept à aligner. En plus, ces noms sont complétés par des synonymes assignés manuellement par l'expert utilisateur. Par ailleurs, le méta-apprenant est aussi entraîné pour générer une matrice définissant un poids pour chaque paire *(concept cible, apprenant de base)*. Pour cela, le méta-apprenant applique la régression linéaire par la méthode des moindres carrés. En effet, étant donné un concept cible c_j, il évalue les poids $W_{A_k}^{c_j}$ qui minimisent la somme suivante :

$$\sum_i (v_i - \sum_k u_i^k * W_{A_k}^{c_j})^2$$

Où A_k désigne le k_{eme} apprenant de base, u_i^k la valeur de similarité proposé par A_k pour le i_{eme} concept source s_i et v_i la vraie valeur de similarité entre s_i et c_j. En fait, v_i prend la valeur 1 si s_i a été manuellement aligné à c_j ou bien 0 sinon. La régression linéaire a pour effet de maximiser les poids des apprenants de base A_k qui proposent des valeurs élevées pour u_i^k lorsque s_i correspond effectivement à c_j et suggèrent des valeurs tendant vers 0 dans le cas contraire. Elle minimise les poids des apprenants ayant un comportement inverse.

Phase d'alignement. Pendant la phase d'alignement, chaque apprenant de base applique le modèle de classification qu'il a établi pendant sa phase d'entraînement,

afin d'estimer les valeurs de similarité de chaque paire *(concept source, concept cible)*. Par exemple, l'apprenant de noms utilise `Whirl` qui est un modèle de classification basé sur la technique du voisin le plus proche développé par Cohen et Hirsh [CH98]. Cet apprenant conserve tous les exemples de paires *(nom de concept source étendu, nom de concept cible)* qu'il rencontre lors de son entraînement. Ensuite, étant donné un concept source s, il détermine le nom de concept cible lui correspondant parmi les noms de tous les exemples stockés dont la distance est inférieur à un seuil prédéfini. La distance entre deux exemples est calculée en se basant sur la métrique TF/IDF (Term frequency/Inverse document frequency) [SM83]. Habituellement, Cette technique est utilisée pour évaluer la pertinence d'un terme par rapport à un document donné en calculant sa fréquence d'apparition dans ce document comparée à sa fréquence dans tout le corpus. Ici, le nom du concept source joue le rôle du terme, et chaque exemple de nom d'un concept cible représente un document. Quant au méta-apprenant, son rôle est de combiner les prédictions des apprenants de base. Pour cela, il utilise les poids qu'il a estimés pour chaque paire *(apprenant de base, concept cible)* pendant la phase d'entraînement. Les résultats combinés sont ensuite confiés à un module qui se charge de leur filtrage. Pour cela, il applique un ensemble de contraintes d'intégrité du domaine définies manuellement par l'expert utilisateur. Une contrainte peut spécifier par exemple le fait qu'un champ cible nommé *nom_employé* ne peut être aligné qu'à un champ source au plus. L'expert utilisateur peut accepter l'alignement après filtrage ou mettre à jour les contraintes pour améliorer le mapping final.

Il est à noter que **LSD** se contente de générer un mapping de type simple (i.e. 1 à 1). Grâce à sa capacité d'exécuter plusieurs modules d'alignement indépendants, on le considère comme un système multi-stratégique.

3.2 iMAP

iMAP [RLD+04] est une extension du système **LSD** [DDH03] supportant la génération des matchings complexes. Rappelons que l'espace de recherche, au niveau

d'une source de données, des combinaisons possibles candidates au matching peut être très large. Pour maîtriser la recherche dans cet espace, **iMAP** emploie un ensemble de modules de recherche chacun étant spécialisé dans la recherche d'un type spécifique de combinaisons. Par exemple, un module de recherche textuel explore les combinaisons qui sont sous forme de concaténations de champs textuels (i.e. concaténations de deux champs et plus parmi tous les champs textuels d'une source de données et en considérant toutes les permutations possibles). L'architecture de **iMAP** se compose des trois unités principales suivantes :

1. *Générateur de combinaisons.* Le générateur de combinaisons reçoit en entrée une source de données `S` et un schéma cible `C` et génère pour chaque concept de `C` les combinaisons candidates de concepts de `S` qui lui correspondent. Cette unité implémente 7 modules de recherche spécialisés respectivement dans (1) la recherche de combinaisons de champs textuels, (2) la découverte de combinaisons de champs numériques, (3) la mise en correspondance entre champs de type catégorie, (4) l'investigation des cas où des données dans la source locale sont modélisées sous forme de concepts dans le schéma cible, (5) la découverte des correspondances sous forme de conversions d'unité de mesure d'un champ source vers un champ cible, (6) la recherche de combinaisons de champs de date et (7) le contrôle de chevauchement entre les données de combinaisons de champs sources avec celles d'un concept cible. Remarquons toutefois que même pour un module de recherche spécialisé, l'espace des combinaisons possibles (e.g. concaténations de champs textuels) peut être considérable. Afin d'assurer une recherche efficace dans les espaces spécialisés, **iMAP** applique une stratégie de recherche basée sur une technique appelée *Beam Search* [NHUTO92]. L'idée de base de cette technique consiste à utiliser une *fonction de classement* permettant d'évaluer les k meilleures combinaisons candidates à chaque niveau d'un arbre de recherche, où k est un paramètre fixé manuellement. La procédure de recherche s'arrête lorsque le rendement global diminue entre deux itérations consécutives de recherche. En fait, **iMAP** garde trace du meilleur score Max_i évalué pour toutes les combinaisons candidates pendant l'itération i et décide de s'arrêter à l'itération $i+1$ si la différence entre Max_{i+1} et

Max_i est inférieure à un certain seuil δ prédéfini. Considérons par exemple, le module spécialisé dans la recherche de combinaisons de champs textuels. Étant donné un champ cible, il commence par évaluer sa similarité avec chaque concept simple de la source locale. Pour cela, il fait appel à un module d'alignement utilisant une approche d'apprentissage basée notamment sur le modèle de `Naïve-Bayes` [DH73] pour la classification des textes. Les concepts sources sont ensuite triés par ordre décroissant de similarité et les k (e.g. k=10) premiers parmi eux sont utilisés pour générer des combinaisons pendant la seconde itération. Pendant cette itération, le module de recherche des combinaisons textuelles forme toutes les concaténations possibles de paires de concepts sélectionnés. Les combinaisons ainsi formées joueront le rôle de nouveaux candidats pour le matching au même titre que les concepts retenus à l'issue de l'itération précédente. L'évaluation, le tri et la sélection des nouveaux candidats sont effectués selon la même procédure que précédemment. Si la condition d'arrêt de la recherche n'est pas vérifiée, une troisième itération est entamée et la recherche continue jusqu'à ce que cette condition soit remplie. Il est à noter que chaque module de recherche spécialisé utilise sa propre stratégie de génération de combinaisons et sa fonction spécifique d'évaluation des candidats au niveau d'une itération. Par exemple, le module de recherche de combinaisons numériques génère des combinaison sous forme d'expressions composées de deux opérandes et un opérateur arithmétique. Pour évaluer les candidats, il s'appuie sur la métrique *Kullback-Leibler* [MS99] qui permet de calculer la similarité entre deux distributions de valeurs.

2. *Estimateur de similarités.* Pendant la phase de génération des combinaisons, les fonctions d'évaluation utilisées par les modules de recherche spécialisés n'exploitent qu'un seul type d'information sémantique pour estimer leurs prédictions respectives. Le rôle de l'estimateur de similarités est de réévaluer les similarités entre les combinaisons générées et les concepts cibles en appliquant une approche multi-stratégique d'alignement exploitant plusieurs type d'information sémantiques en même temps. Pour cela, **iMAP** s'appuie sur le système **LSD**. L'utilisation de **LSD** pendant la procédure de recherche de combinaisons a été évitée car elle aurait eu

un impact important sur le temps de son exécution.

3. *Sélecteur de matchings.* Le sélecteur de matchings se charge de la génération du mapping final à partir des alignements proposés par l'estimateur de similarités. Plus précisément, il procède à l'élimination des combinaisons invalides et trie les paires (concept source, concept cible) pour ne retenir que celles dont la valeur de similarité dépasse un certain seuil prédéfini. Le sélecteur de matchings valide les combinaisons de concepts en ayant recours à une base de connaissances du domaine alimentée manuellement par l'utilisateur expert. Cette base stocke entre autres les contraintes d'intégrité liées aux sources de données à aligner et des règles de gestion. Une règle de gestion peut stipuler par exemple que les concepts *nom_employé* et *prix_produit* ne doivent pas figurer ensemble dans une même combinaison puisqu'il n'existe aucune correlation sémantique entre eux.

En somme, **iMAP** est un système multi-stratégique générant des mappings simples et complexes entre des sources de données locales et un schéma central. Il est toutefois à préciser que les matchings complexes générés par **iMAP** se limitent à ceux de cardinalité *n à 1*. En effet, les modules spécialisés de recheche génère des combinaisons pour chaque champ simple du schéma cible uniquement.

3.3 COMA++

Le système **COMA++** [DAR05, DR02] est une plateforme générique pour le mapping entre sources de données structurées. Il implémente une librairie extensible de fonctions d'alignement. Il permet à l'utilisateur de sélectionner les fonctions à exécuter lors du processus d'alignement. Il permet également de combiner les résultats de ces fonctions de façon flexible pour générer le mapping final. Parmi les formats de modèles supportés par **COMA++**, on trouve le format XSD, XDR (XML Data Reduced), OWL et les schémas relationnels. En fait, l'architecture de **COMA++** comporte un module appelé *Schema Pool* qui reçoit en entrée les modèles de données à aligner qu'il convertit sous forme de graphes acycliques dirigés. Tous les modèles convertis sont stockés dans une base de données en interne. Cette base contient aussi des informations auxiliaires

utiles pour le mapping telles que des taxonomies spécifiques à certains domaines donnés et des tables de synonymes. Par ailleurs, l'architecture de **COMA++** inclut aussi un module appelé *Match Customizer* permettant de configurer une combinaison de fonctions d'alignement, un moteur d'exécution responsable de la génération du mapping et enfin un module appelé *Mapping Pool* se chargeant de la sauvegarde d'un mapping généré dans la base de données ou de son exportation pour une utilisation externe. Il est à noter que le module *Mapping Pool* offre également différentes fonctions pour la manipulation des mappings générés telles que la fusion ou la composition transitive de mappings.

Le processus d'alignement de **COMA++** s'exécute en une ou plusieurs itérations. Chaque itération est composée des trois étapes suivantes :

1. *Étape 1* : Pendant cette étape l'utilisateur intervient pour choisir manuellement la ou les fonctions d'alignement à exécuter. Il définit également la stratégie pour combiner leurs résultats respectifs (e.g. moyenne, maximum, etc.).
2. *Étape 2* : Cette étape consiste en l'exécution des fonctions de matching choisies. Chaque fonction génère son propre mapping sous forme d'un ensemble de triplets (élément source, élément cible, similarité). Si l'on a k fonctions sélectionnées, m éléments sources et n éléments cibles, le résultat de cette étape sera un cube de valeurs de similarités de dimension $k \times m \times n$.
3. *Étape 3* : Au cours de cette étape, le moteur d'exécution se charge de la combinaison des mappings générés pendant l'étape 2 en appliquant la stratégie de combinaison définie dans l'étape 1. Le mapping combiné peut être validé par l'utilisateur expert en sélectionnant les triplets dont les valeurs de similarité dépassent un certain seuil prédéfini ou bien raffiné en revenant à l'étape 1 pour recommencer une nouvelle itération.

Il est à remarquer que l'intervention de l'utilisateur fait partie intégrante du processus d'alignement de **COMA++**. En particulier, la qualité du mapping final dépend fondamentalement des choix faits par l'utilisateur expert des stratégies de matching et de leur combinaison. Le mapping généré par **COMA++** est de cardinalité *un à un*. Étant donné qu'il permet d'exécuter plusieurs fonctions de matching indépendantes pendant

un même processus d'alignement, **COMA++** appartient à la classe des systèmes multi-stratégiques.

3.4 V-Doc

V-Doc [QHC06] est un système d'alignement d'ontologies. Les sources de données à aligner y sont présentées sous forme de graphes dirigés basés sur le modèle RDF. **V-Doc** construit un document virtuel pour chaque noeud des graphes à aligner. Chaque noeud représente un concept dont le document virtuel est composé de mots extraits de son nom ainsi que des commentaires et annotations qui lui sont associés au sein d'une ontologie. **V-Doc** évalue chaque paire de noeuds en comparant les documents virtuels qui leur ont respectivement été associés. Ainsi, le processus global de mapping de **V-Doc** s'exécute selon les deux étapes suivantes :

1. *Construction des documents virtuels* : Rappelons qu'un graphe RDF est composé d'un ensemble de triplets composés d'une `ressource`, d'une `propriété` et d'une `valeur`. Pour chaque noeud du graphe **V-Doc** construit une *description* en fonction de la nature de l'élément qu'il représente. En particulier, lorsqu'il s'agit d'une valeur, la description créée consiste en une collection de mots extraits directement de la valeur elle-même. Par ailleurs, dans le cas d'un noeud `N` représentant une ressource ou bien une propriété identifiée par une référence URI, la description construite consiste alors en une collection de mots dérivés du nom local, des balises `rdfs :label` et `rdfs :comment`, ainsi que d'autres annotations possibles. Dans une description les mots extraits sont pondérés en fonction de leur origine, comme le suggère la formulation suivante :

$$DES(N) = \alpha_1 * CollectionDeMotsDansNomLocal(N)$$
$$+\alpha_2 * CollectionDeMotsDansLabel(N)$$
$$+\alpha_3 * CollectionDeMotsDansComment(N)$$
$$+\alpha_4 * CollectionDeMotsDansAnnotations(N)$$

où α_1, α_2, α_3 et α_4 sont des coefficients rationnels compris dans l'intervalle $[0, 1]$. Ils représentent les poids associés aux catégories de mots formant la description de N. Le signe + est un opérateur de fusion de deux collections.

Soient $R(N)$, $P(N)$ et $V(N)$ les ensembles de voisins de N lorsque celui-ci représente respectivement une ressource, une propriété ou une valeur dans un triplet au sein d'une ontologie. Nous avons $R(N) = \bigcup_{ress(t)=N}\{prop(t), val(t)\}$, où $ress(t)$, $prop(t)$ et $val(t)$ désignent respectivement la ressource, la propriété et la valeur d'un triplet t donné. $P(N)$ et $V(N)$ sont définis de façon analogue à $R(N)$. La définition du document virtuel associé à N est donnée par la formule suivante.

$$DV(N) = DES(N)$$
$$+\gamma_1 * \sum_{N' \in R(N)} DES(N')$$
$$+\gamma_2 * \sum_{N' \in P(N)} DES(N')$$
$$+\gamma_3 * \sum_{N' \in V(N)} DES(N')$$

où γ_1, γ_2 et γ_3 sont des coefficients rationnels compris dans l'intervalle $[0, 1]$. Ils servent à pondérer respectivement les ensembles $R(N)$, $P(N)$ et $V(N)$.

2. *Comparaison des documents virtuels* : Un document virtuel est une collection de mots pondérés. Il peut donc être représenté par un vecteur dont les composants sont les poids attribués aux mots qu'il contient. **V-Doc** utilise ainsi le modèle vectoriel (i.e. Vector Space Model) [RW86] pour calculer la similarité entre les documents virtuels. Il applique notamment la technique TF/IDF [SM83] qui est communément utilisée dans le domaine de *recherche d'information*, pour comparer deux documents dans un espace vectoriel de dimension N, où N est le nombre total de mots distincts présents dans les documents virtuels appartenant à cet espace. Pour chaque mot dans un document virtuel donné, **V-Doc** évalue un score égal au produit des deux coefficients TF et IDF. Ces derniers sont calculés en utilisant

les formules suivantes.

$$TF = \frac{w}{W}$$

$$IDF = \frac{1}{2} * (1 + \log_2)\frac{N}{n}$$

où w désigne le poids du mot au sein d'un document et W la somme des poids de tous les mots composant ce document. Par ailleurs, n est le nombre de tous les documents virtuels contenant ce mot et N représente le nombre total de documents. Ainsi, le score attribué à un mot dans un document donné reflète son degré de pertinence spécifiquement pour ce document. La similarité entre deux documents virtuels est enfin évaluée en comparant les vecteurs $\overrightarrow{V_i}$ et $\overrightarrow{V_j}$ qui les représentent respectivement. Pour cela, **V-Doc** a recours à la métrique *Cosine* [EBB+04] évaluée en appliquant la formule suivante.

$$cos(\overrightarrow{V_i}, \overrightarrow{V_j}) = \frac{\sum_{k=1}^{N} p_{ik} p_{jk}}{\sqrt{\sum_{k=1}^{N} p_{ik}^2 \sum_{k=1}^{N} p_{jk}^2}}$$

où p_{ik} et p_{jk} désignent respectivement les k_{eme} composants des vecteurs $\overrightarrow{V_i}$ et $\overrightarrow{V_j}$. Ainsi, dans le cas limite où les deux documents ne partagent aucun mot, la similarité sera nulle. Si au contraire tous les poids des mots qui les composent respectivement sont équivalents, la similarité sera de 1.

Le système **V-Doc** repose donc sur une approche de mapping linguistique pour aligner des ontologies. Sa stratégie de matching se base sur l'exploitation des ressources lexicales liées aux concepts dans le but de déduire des correspondances sémantiques entre eux. **V-Doc** fait partie des systèmes mono-stratégiques générant des mapping simples entre sources de données.

3.5 Cupid

Cupid [JPE01] est un algorithme générique pour l'alignement entre schémas de sources de données. Il emploie un modèle arborescent pour représenter en interne les sources de données à aligner. Dans le cas où la structure d'un schéma n'est pas

arborescente, il choisit un noeud comme racine et génère tous les chemins possibles allant de ce noeud vers les autres noeuds de façon à ce que le modèle final généré soit un arbre. L'algorithme **Cupid** est donc spécialisé dans l'alignement entre deux graphes arborescents. La valeur de similarité calculée pour chaque paire de noeuds tient compte à la fois de leur *similarité linguistique* et de leur *similarité structurelle* :

1. `Similarité linguistique` : **Cupid** calcule une similarité linguistique pour chaque paire de noeuds en faisant le rapprochement entre les noms des concepts respectifs qu'ils représentent. Pour ce faire, les noms de concepts des deux graphes sont tous normalisés. La normalisation d'un nom consiste à le décomposer en des sous-noms en se basant notamment sur la ponctuation ou les lettres majuscules, à résoudre les éventuelles abréviations qu'il peut contenir et à éliminer les mots inutiles (e.g. articles, prépositions, etc.). Le recours à un dictionnaire ou à un thésaurus externe est ici nécessaire. **Cupid** classifie ensuite tous les concepts en des catégories définies par leurs types de données respectifs. Enfin, un coefficient de similarité linguistique, noté *lsim*, est calculé pour chaque paire de noeuds. La valeur de *lsim* est calculée en utilisant une métrique de comparaison lexicale basée sur l'emploi d'un dictionnaire décrivant des relations de synonymie et d'hypernimie entre concepts appartenant à un même domaine d'application. **Cupid** ne compare que les concepts d'une même catégorie.
2. `Similarité structurelle` : L'évaluation de la similarité structurelle repose sur le principe selon lequel deux noeuds sont similaires si leurs voisinages respectifs le sont aussi et vice versa. Dans une phase initiale, une valeur de similarité est attribuée à chaque paire de feuilles. Cette valeur correspond au degré de compatibilité entre les types de données associées à chacune des deux feuilles. Cette valeur de similarité est directement lue dans une table de valeurs de compatibilité définie par l'expert utilisateur. Ces valeurs sont comprises entre 0 et 0.5 afin qu'elles puissent être augmentées au cours de la phase suivante (i.e. pendant la procédure du calcul itératif des similarités structurelles). Parcourant les deux arbres à aligner des feuilles vers la racine, cette procédure calcule la similarité structurelle pour chaque paire de noeuds en deux étapes. (1) Tout d'abord, cette similarité est initialisée

par la valeur mesurant la proportion des paires de feuilles fortement liées des deux sous-arbres dont les racines représentent respectivement les deux noeuds en cours de comparaison. Dans le cas où les noeuds sont tous les deux des feuilles cette valeur correspond à celle de la similarité linguistique calculée lors de la phase initiale. Deux feuilles sont considérées fortement liées si la valeur de leur similarité pondérée $wsim = w_{struct} \times ssim + (1 - w_{struct}) \times lsim$ dépasse un certain seuil S_{accept} prédéfini par l'expert. La valeur de similarité structurelle entre les deux noeuds est donnée par la formule suivante :

$$ssim(s,t) = \frac{\left| \begin{array}{l} \{x | x \in feuilles(s) \wedge \exists y \in feuilles(t), lienfort(x,y)\} \\ \cup \{x | x \in feuilles(t) \wedge \exists y \in feuilles(s), lienfort(x,y)\} \end{array} \right|}{|feuilles(s) \cup feuilles(t)|}$$

où $feuilles(s)$ désigne l'ensemble des feuilles du sous-arbre dont la racine est le concept s et $lienfort(x,y)$ est un prédicat exprimant que les concepts x et y sont fortement liés. (2) Ensuite, si la valeur de similarité pondérée entre les noeuds en cours de comparaison dépasse un certain seuil S_{haut} prédéfini, la valeur de similarité de chaque paire de feuilles des deux sous-arbres chapeautés par ces noeuds est augmentée par un facteur c_{inc} prédéfini. Inversement, si la valeur de similarité pondérée est plus petite qu'un certain seuil S_{bas} la similarité entre les feuilles des sous-arbres est diminuée par un certain facteur c_{dec}.

La similarité finale entre deux noeuds est obtenue par pondération des valeurs de similarité linguistique et structurelle selon la formule suivante $wsim(s,t) = w_{struct} \times ssim(s,t) + (1 - w_{struct}) \times lsim(s,t)$, où w_{struct} est une constante fixée entre 0 et 1. Le mapping final est obtenu en sélectionnant toutes les paires de noeuds (s,t) satisfaisant la condition $wsim(s,t) > S_{accept}$, où S_{accept} est un seuil prédéfini. **Cupid** combine donc les résultats de deux calculs de similarité qu'il effectue au sein d'un même algorithme. De ce fait, son approche est dite hybride. Les concepts étant comparés un à un, le mapping généré par **Cupid** est dit simple.

3.6 SF

SF (Similarity Flooding) [MGMR02] est un algorithme générique pour l'alignement entre divers modèles de données. Afin de ne pas être dépendant d'un format particulier, **SF** convertit d'abord à l'interne tous les modèles sous forme de graphes dirigés étiquetés. Les noeuds des graphes ainsi générés peuvent représenter aussi bien des informations liées aux schémas que des instances de données. Dans un graphe, chaque arc est décrit par un triplet (x, p, y), où x et y sont respectivement les noeuds de départ et d'arrivée alors que p dénote l'étiquette portée par l'arc. Considérons par exemple la représentation sous forme d'un graphe d'une source de données relationnelle. Les tables et les champs sont représentés par des noeuds. Un noeud portant l'identificateur d'un champ est lié à un autre noeud libellé *champ* via un arc portant l'étiquette *type*, un autre arc portant l'étiquette *champs* est tiré de ce noeud vers un autre noeud identifiant une table, etc. Considérons maintenant l'alignement de deux graphes G_1 et G_2. Pour évaluer les similarités entre les noeuds composant respectivement G_1 et G_2, l'algorithme **SF** raffine itérativement ses estimations de similarité jusqu'à l'obtention d'un point fixe. Il se base sur l'intuition selon laquelle deux noeuds sont similaires si et seulement si leurs noeuds adjacents le sont aussi. L'algorithme **SF** s'exécute en trois étapes comme suit :

1. *Construction du graphe de propagation de similarités* : À partir de deux graphes G_1 et G_2, **SF** crée tout d'abord un *graphe de connectivité* (GC) où chaque noeud représente une paire de noeuds (x, y) dont les composantes appartiennent respectivement à G_1 et G_2. Les arcs de GC sont tels que $((x, y), p, (x', y')) \in arcs(GC(G_1, G_2))$ $\Leftrightarrow$ $(x, p, x') \in arcs(G_1) et (y, p, y') arcs(\in G_2)$. Donc, seuls les arcs portant des étiquettes équivalentes respectivement dans les deux graphes sont pris en compte dans la construction d'un GC. Deux noeuds reliés directement par un arc sont appelés *noeuds voisins*. Le GC n'étant pas dirigé, pour chaque arc $((x, y), p, (x', y'))$ du GC, un autre arc de sens opposé (i.e. $((x', y'), p, (x, y))$) est créé. Enfin, des poids entre 0 et 1 sont attribués à chacun des arcs composant GC. Chaque poids représente la part de similarité à faire propager entre deux noeuds. Pour cette raison, les poids sont appelés *coefficients de propagation*. Les valeurs initiales des coefficients de pro-

pagation sont réparties sur les arcs selon la règle suivante : pour tout noeud (x, y) du graphe GC, un poids de valeur $\frac{1}{n}$ est attribué à chaque arc sortant de (x, y), où n est le nombre total d'arcs sortants de (x, y). Le graphe GC muni des coefficients est appelé *graphe de propagation* GP.

2. *Procédure de calcul d'un point fixe* : soit σ une fonction de calcul de similarité. L'application de σ sur l'ensemble des noeuds $(x, y) \in G_1 \times G_2$ du graphe (GP) construit pendant la phase précédente permet d'évaluer la similarité des noeuds de G_1 et G_2. L'algorithme **SF** se base sur un calcul itératif des valeurs $\sigma(x, y) \geq 0$. Notons σ^i le mapping généré pendant l'itération i. **SF** commence par calculer un mapping initial σ^0 en appliquant une métrique de comparaison lexicale sur tous les noeuds de GP. À chaque itération, la valeur σ de chaque noeud de GP est augmentée par la somme pondérée des valeurs σ de tous les noeuds liés à lui par un arc entrant. La formule suivante exprime l'opération de propagation des similarités appliquée à un noeud (x, y) de GP à l'itération $i + 1$.

$$\sigma^{i+1}(x, y) = \sigma^i(x, y) + \sum_{((x', y'), p, (x, y)) \in arcs(GP)} \sigma^i(x', y') * p$$

Une fois réévaluées, toutes les similarités sont normalisées en les divisant par la valeur maximale de toutes les similarités calculées pendant une itération. Le calcul itératif continue jusqu'à l'itération n telle que la distance euclidienne entre les vecteurs σ^n et σ^{n-1} ne dépasse pas une valeur positive ε prédéfinie.

3. *Filtrage* : Le filtrage du mapping généré pendant la phase précédente s'effectue en deux étapes. Dans un premier temps, certaines contraintes spécifiques sont appliquées. Par exemple, **SF** peut décider de ne garder que les paires de concepts de même type (e.g. rejeter les cas où des colonnes ont été alignées à des tables). Dans un second temps, **SF** calcule une métrique pour chaque paire de concepts afin de sélectionner le mapping final. Typiquement, un seuil de similarité est employé pour ne retenir que les paires de concepts ayant une valeur de similarité dépassant ce seuil.

En somme, **SF** est un algorithme qui exploite en parallèle des informations sur les schémas et sur les instances de données pour générer un mapping. Il se contente de générer un mapping simple (*1 à 1*). Étant donné que **SF** implémente une seule procédure d'alignement, on le considère comme un système mono-stratégique.

3.7 Classification des systèmes

Nous proposons une classification des systèmes étudiés basée sur les quatre critères suivants :

1. *Modèles de données supportés* : Tandis que certains systèmes d'alignement se spécialisent dans le mapping entre sources de données décrites dans un modèle spécifique, d'autres sont capables d'aligner divers types de modèles. Parmi les modèles qui sont habituellement supportés par les solutions actuelles on peut citer le modèle relationnel (généralement décrit dans le langage SQL), les schémas liés au format XML (i.e. DTD ou XSD) et les ontologies qui sont généralement décrites dans le langage RDF ou OWL. Pour intégrer plusieurs types de modèles, un système d'alignement s'appuie normalement sur un ensemble de modules externes, chacun étant spécialisé dans la traduction d'un type de modèle spécifique. Typiquement, les sources de données sont converties sous forme de graphes internes dont les noeuds représentent les éléments sémantiques (e.g. noms de concepts, types de données, etc.) qui les composent et les arêtes expriment les relations entre ces éléments (e.g. relation de généralisation dans le cas d'une ontologie).
2. *Types d'informations sémantiques exploitées* : pour établir des correspondances sémantiques entre des paires de concepts, les systèmes d'alignement exploitent deux catégories principales d'informations sémantiques les caractérisant. (1) La première catégorie se compose des informations liées aux schémas des sources de données à aligner. Ces informations concernent les caractéristiques lexicales et syntaxiques liées aux concepts qui les composent respectivement. Elles incluent par exemple les noms des concepts, leurs types de données et leurs structures. (2) La seconde catégorie concerne les instances de valeurs associées aux concepts. Plusieurs tech-

niques de comparaison de contenus de concepts peuvent être utilisées selon leurs types de données respectifs. Par exemple, l'appariement de deux séries de valeurs numériques peut se baser sur la comparaison de certaines de leurs propriétés statistiques (e.g. moyenne, médiane, etc.). Ou encore, pour tenter de rapprocher deux contenus textuels, la comparaison des mots les plus fréquents peut être utilisée.

3. *Approche d'alignement adoptée* : l'approche d'un système d'alignement peut être mono-stratégique, hybride ou multi-stratégique. (1) Une approche mono-stratégique emploie une seule stratégie de mapping. Cette stratégie peut exploiter un ou plusieurs types d'informations sémantiques liées aux sources de données pour déduire l'alignement entre elles. (2) Une approche hybride combine de façon rigide deux ou plusieurs stratégies de matching. Elle les exécute séquentiellement à l'intérieur d'un processus unique d'alignement et combine leurs résultats respectifs pour générer le mapping final. (3) Enfin, une approche multi-stratégique exécute deux ou plusieurs stratégies d'alignement indépendantes et combine leurs résultats en espérant maximiser la précision du mapping final. Dans une telle approche, chaque stratégie individuelle est implémentée par un module séparé qui peut à lui seul être utilisé pour aligner deux sources de données.
4. *Type de matchings généré* : les matchings générés sont soit de type simple (1 à 1) ou bien de type complexe (1 à n, n à 1 et n à m).

Ces critères mettent en évidence la flexibilité d'un système d'alignement des données. C'est la raison pour laquelle nous les avons choisis pour classifier les solutions étudiées. Un algorithme générique de mapping supportant plusieurs modèles de sources de données présente l'avantage d'être plus facilement intégré dans une solution tierce d'alignement plutôt qu'un module spécialisé dans le traitement d'un seul modèle spécifique. Par ailleurs, il semble naturel que plus un système exploite un nombre important d'informations sémantiques pour déduire un alignement entre des sources, plus il sera précis. Le type d'approche adoptée par un système d'alignement peut aussi influencer sa performance. En général, l'emploi de plusieurs stratégies de matching en parallèle permet de traiter différents types d'informations sémantiques séparément et par conséquent de

TAB. 3.1 – Classification des systèmes d'alignement

	Modèle de données			Informations sémantiques		Approche			Type de matching	
	Modèle relationnel	XML	Ontologie	Schémas	Instances	Mono-stratégique	Hybride	multi-stratégique	simple	complexe
LSD		×		×	×			×	×	
iMAP		×		×	×			×	×	×
COMA++	×	×	×	×	×			×	×	
V-Doc			×	×	×	×			×	
Cupid	×	×	×	×			×		×	
SF	×	×	×	×		×			×	

mieux maîtriser leur analyse. Il est à noter par ailleurs qu'un système basé sur une approche multi-stratégique se distingue par sa qualité d'extensibilité. En effet, l'intégration d'une nouvelle stratégie de matching ne nécessite alors qu'une simple mise à jour d'un module responsable de combiner ses résultats avec d'autres stratégies existantes. Enfin, un système capable de générer les matchings complexes explore l'espace total des paires de matching possibles alors qu'un système se limitant aux matchings simples se contente souvent de générer des mappings partiels. Les matchings complexes sont pourtant fréquents en pratique.

La table 8.4 présente une classification des systèmes étudiés selon les critères susprésentés. Elle montre que les systèmes **COMA++**, **Cupid** et **SF** supportent tous les modèles de données usuels. En revanche, certains systèmes se limitent au traitement d'une représentation spécifique des données, tels que **LSD** et **V-Doc** qui sont respectivement spécialisés dans l'alignement des modèles XML et des ontologies. Même si ces systèmes ont l'inconvénient d'offrir des solutions ad hoc, ils ont souvent l'avantage d'être relativement plus performants quand il s'agit d'aligner des modèles relevant de leur spécialité. Par ailleurs, nous constatons que la moitié des systèmes étudiés reposent sur une approche multi-stratégique, en l'occurrence **LSD**, **iMAP** et **COMA++**. Notons que **COMA++** se distingue par une flexibilité supérieure concernant la configuration de stratégies de matching dans un processus d'alignement. En effet, il met à la disposition de l'utilisateur une librairie extensible de fonctions d'alignement et de combinaison de résultats de mapping, à partir de laquelle il peut sélectionner les fonctions à exécuter pen-

dant chaque étape d'un processus itératif d'alignement. À l'exception de **Cupid** et **SF**, tous les systèmes exploitent par ailleurs, les informations liées aux schémas de sources de données et les instances de valeurs associées à leurs concepts pour élaborer le mapping entre elles, . En effet, ces derniers se limitent à utiliser les ressources d'information schématiques. Ceci leur confère par contre la souplesse d'être utilisés dans un nombre considérable d'applications d'alignement où les sources de données se présentent sous forme de simples schémas. Enfin, **iMAP** se distingue par le fait d'être le seul parmi les systèmes étudiés à être capable de générer des mappings complexes.

3.8 Conclusion

La classification sus-présentée met en évidence les qualités importantes liées aux systèmes d'alignement existants. Elle constitue aussi un moyen simple et efficace pour identifier rapidement les principales lacunes d'une application de matching. Le chapitre qui suit expose les limitations des solutions actuelles et trace ensuite les grandes lignes de l'approche que nous proposons pour résoudre le problème du mapping sémantique.

CHAPITRE 4

INDIGO : UNE APPROCHE MULTI-STRATÉGIQUE ADAPTATIVE

Notre approche s'inspire d'une analyse critique des systèmes actuels de mapping sémantique de données dont un échantillon a été présenté dans le chapitre précédent. Elle propose une solution globale d'alignement de données permettant d'en combler certaines lacunes et d'améliorer leurs techniques d'appariement. Ce chapitre comporte deux sections : la première présente les limitations des systèmes actuels ; la seconde, décrit l'approche INDIGO que nous proposons.

4.1 Limitations des systèmes actuels

Au cours de l'élaboration de l'état de l'art, nous avons relevé trois aspects de la problématique d'alignement des données pour lesquels il n'existe pas encore de solution viable ou qui méritent d'être explorés plus en profondeur :

- La qualité d'un mapping (i.e. précision et rappel) est affectée par l'insuffisance des informations sémantiques dont les ressources, pour leur extraction, se limitent le plus souvent aux schémas et instances des sources de données à aligner.
- Les approches de mapping actuelles n'ont pas été conçues pour s'adapter automatiquement au changement des données. En particulier, les systèmes multi-stratégiques n'ont pas encore exploré les combinaisons dynamiques des stratégies individuelles qu'ils implémentent en fonction des caractéristiques des sources de données à aligner.
- Il n'y a que peu de solutions qui supportent le mapping complexe. Ce dernier posant un problème difficile à résoudre dû au fait que l'espace de recherche de combinaisons de concepts possibles est très large.

4.1.1 Insuffisance des ressources sémantiques

Pour élaborer un mapping, un système d'alignement s'appuie sur les informations sémantiques (e.g. type de données, distribution de valeurs d'un champ numérique, etc.) annotant les concepts à aligner. Sa performance dépend donc directement de la qualité et de la quantité de ces informations qui, aujourd'hui, sont généralement déduites à partir de l'analyse des schémas de sources de données (**XSD**, **DTD**, **RDFS**, etc.) ou bien des valeurs liées aux concepts qui les composent respectivement. Or, se limiter à ces deux sortes d'informations seulement s'avère en pratique insuffisant pour lever les ambiguïtés empêchant d'apparier correctement les concepts sémantiquement similaires. Voici quelques exemples typiques des ambiguïtés concernant les noms de concepts.

- Le nom de l'attribut *Commande :date1* (de l'entité *Commande*) présente des informations peu précises concernant l'entité sémantique qu'il représente : s'agit-il d'une date de facture, une date de livraison ou d'une date de réception ?
- l'attribut *Employé :nom* ne spécifie pas s'il représente le nom complet d'un employé ou bien juste son prénom.
- Le nom *Facture :montant* n'indique pas si les valeurs qui lui sont associées incluent ou non la taxe.
- L'attribut *Commande :lc* utilise une forme abbrégée qui ne permet pas de déterminer facilement, voire automatiquement, qu'il s'agit d'une *ligne de commande.*

En général, pour qu'elles soient performantes, les stratégies d'alignement doivent pouvoir recourir à des informations sémantiques pertinentes et suffisantes . Dans le cas particulier des stratégies basées sur la comparaison lexicale de noms de concepts, l'expérience a montré que leur efficacité diminue quand il s'agit d'aligner des noms présentant les caractéristiques suivantes :

- Les noms ne possèdent pas de synonymes clairs et précis tels que les noms *commentaire* et *plan.*
- Ils désignent des concepts bien spécifiques mais sont déclarés sous une forme plutôt réduite. Par exemple, le nom *domicile* dans une source de données peut en fait représenter *le numéro de téléphone à domicile.*

- Ils portent une sémantique large, voire abstraite, qui peut correspondre à une panoplie de concepts différents tels que les noms *chose* ou *entité*.

De façon similaire, une stratégie d'alignement de contenus de concepts, basée par exemple sur la comparaison de distribution d'unités lexicales, s'avérerait inappropriée pour aligner des champs numériques.

4.1.2 Limitation de la flexibilité des approches multistratégiques actuelles

Les systèmes basés sur une approche multi-stratégique gagnent de plus en plus de popularité au sein des communautés de recherche opérant dans le domaine du mapping sémantique des données. Certaines solutions actuelles, telles que celles proposées par **LSD** [DDH03] et **COMA++** [DAR05, DR02], ont déjà prouvé leur qualité en terme d'efficacité et de flexibilité architecturale. Toutefois, elles restent des expériences ad hoc et manquent encore d'adaptabilité. Leurs concepteurs n'ont pas exploré, par exemple, les techniques de sélection dynamique de stratégies de matching à appliquer en fonction des caractéristiques des sources de données à aligner. Dans le cas du système **LSD**, une technique appelée Stacking [Wol92, TW99] est utilisée par le méta-apprenant pour estimer les poids des apprenants de base après avoir testé leurs performances respectives lors d'une phase préalable d'entraînement visant à aligner chaque élément de la source de données cible. Toutefois, l'estimation de ces poids est effectuée en supposant l'alignement vers un schéma cible fixe qui est le schéma central dans lequel seront intégrées les sources locales. Si le schéma central est modifié, les poids devront alors être réestimés et par conséquent d'autres données d'entraînement devront être élaborées. Quant à **COMA++**, il offre une plateforme supportant plusieurs algorithmes de matching ainsi que différentes méthodes pour les combiner. Cependant, il ne propose pas de mécanisme permettant une sélection dynamique des algorithmes de matching, ni des méthodes destinées à les combiner. C'est l'utilisateur qui les choisit manuellement et demeure ainsi le principal stratège dans le processus d'alignement. Par conséquent, l'adaptabilité de la solution de mapping proposée par **COMA++** reste tributaire de l'expertise spécifique de l'utilisateur.

En somme, même si les systèmes multi-stratégiques actuels ont franchi un pas important vers une plus grande flexibilité, ils n'ont pas encore atteint le niveau d'adaptabilité

requis par le web. En effet, l'expansion d'internet ouvre la voie vers une diversité toujours croissante de sources de données. Les développeurs seront de plus en plus confrontés à des problèmes liés aux applications d'intégration des données, de réconciliation de schémas et d'alignement d'ontologies. Des solutions très spécifiques aux problèmes d'alignement risquent de devenir rapidement obsolètes ou non utilisables. Les futurs systèmes devront être dotés d'une grande capacité d'adaptation afin de faire face à une diversité accrue de problèmes d'alignement et ce, tout en maintenant un bon niveau de performance.

4.1.3 Problème du mapping complexe

Actuellement, très peu de systèmes supportent le mapping complexe. Le focus a surtout été mis sur le problème du mapping simple qui est relativement plus facile à résoudre. En effet, l'élaboration d'un mapping complexe implique la recherche, au niveau de chaque source de données à aligner, de toutes les combinaisons de concepts possibles candidates au matching. Or, en pratique, l'espace de recherche de ces combinaisons est très large ; ce qui rend son exploration très difficile et peu efficace. Le système **iMAP** [RLD+04] fait partie des premières initiatives ayant relevé le défi d'automatiser le processus de génération du mapping complexe. Comme déjà mentionné, **iMAP** est basé sur plusieurs modules de recherche, chacun étant spécialisé dans l'exploration d'un sous-espace de combinaisons de concepts. Toutefois, ces sous-espaces (e.g. sous-espaces de combinaisons arithmétiques ou textuelles) peuvent eux-même en pratique, être très larges. Pour contrôler sa recherche dans ces ensembles de combinaisons, **iMAP** utilise une technique standard de recherche appelée *Beam Search* [NHUTO92] dont le principe est de limiter l'espace de recherche seulement aux concepts participant aux k meilleurs matchings avec la source de données cible, et ce, à chaque itération de recherche. Le paramètre k est prédéfini arbitrairement (c.f. section 3.2). Même si **iMAP** constitue une avancée intéressante dans le domaine du calcul de mappings complexes, il présente encore certaines limitations. Tout d'abord, ce système se concentre sur la découverte des matchings de cardinalité *n à 1* uniquement. En effet, les combinaisons de concepts sont recherchées du côté de la source de données de départ seulement, ce qui empêche de découvrir les alignements de cardinalités *1 à n* et *n à n*. Ce choix a vraisemblablement

été fait pour limiter le coût de la procédure de recherche de combinaisons qui autrement aurait été trop élevé. Par ailleurs, l'algorithme de recherche itérative *Beam Search*, tel qu'implémenté par **iMAP**, peut présenter des performances de recherche insuffisantes en terme de rappel. En effet, le seuil fixant le nombre maximum d'itérations de recherche présente le risque potentiel d'empêcher la découverte de plusieurs combinaisons pertinentes en stoppant la recherche trop tôt. Inversement, en générant systématiquement des combinaisons aléatoires au niveau de chaque itération, **iMAP** peut aussi manquer de précision. En effet, certaines combinaisons parasites peuvent possiblement être générées à un stade avancé de la recherche et se voir sélectionnées à la place d'autres qui sont effectivement pertinentes. Typiquement, ceci arrive lorsqu'une mauvaise combinaison, présentant des caractéristiques sémantiques similaires à celles d'une bonne combinaison, est analysée par l'algorithme de recherche avant cette dernière. Considérons par exemple, les deux combinaisons suivantes *(prix +1)*rabais* et *(prix+1)*taxes*. Si les contenus respectifs des attributs numériques *rabais* et *taxes* présentent des propriétés statistiques similaires alors ces deux combinaisons seraient aussi similaires. Or, si pendant la recherche d'un candidat source correspondant au concept cible *prix_toute_taxe_comprise*, **iMAP** génère la combinaison *(prix +1)*rabais* en premier, alors il la retiendra à la place de *(prix+1)*taxes* qui est la bonne.

4.2 L'approche INDIGO

Notre approche propose une solution globale aux problèmes sus-énumérés en définissant une architecture de système de mapping qui est à la fois flexible et adaptable. L'adaptabilité d'une application d'alignement de données peut être définie comme sa capacité à tenir compte 1) du contexte (facteurs externes) et 2) des caractéristiques (facteurs internes) des sources de données à aligner pour influencer l'exécution de son processus de matching en vue d'améliorer ses performances globales.

- **Intégration du contexte :** deux noms différents peuvent parfois désigner un même concept, comme il est possible qu'un seul même nom porte des sens différents selon le contexte où il est utilisé. La synonymie et l'hypernymie consituent les deux

sources majeures d'ambiguïtés rencontrées lors d'un processus de matching. Par exemple, un mot tel que *fréquence* porte des sens différents selon qu'il est employé dans le domaine de statistiques ou dans celui de la radio-diffusion. La spécification du contexte est souvent nécessaire pour cerner la sémantique d'un concept. Nous pensons que le fait de lier les informations contextuelles aux concepts à aligner peut aider un système de mapping à améliorer aussi bien sa précision que son rappel.

- **Prise en compte des caractéristiques de sources de données :** L'efficacité des stratégies d'alignement dépend de la nature des informations sémantiques liées aux concepts à aligner. Certaines stratégies de matching peuvent être mieux adaptées pour exploiter certaines formes spécifiques d'informations que d'autres. Par exemple, parmi les stratégies d'alignement spécialisées dans la comparaison des noms de concepts, celles qui utilisent des techniques de desabréviation basées notamment sur des thésaurus spécialisés seraient les plus appropriées dans le cas où les noms à comparer se présentent sous des formes abrégées. Les stratégies basées sur des techniques d'apprentissage, ou encore celles utilisant des fonctions de comparaison lexicale ayant recours à des dictionnaires génériques (e.g. WordNet), seraient moins performantes dans ce cas.

Notre nouvelle approche INDIGO est inspirée des deux observations sus-mentionnées. Elle propose une architecture présentant une meilleure flexibilité que celles des systèmes de matching actuels. Elle offre également une solution de mapping adaptable et plus performante tenant compte du contexte des sources de données à aligner. Nous préconisons une architecture composée de deux modules : un *analyseur de contexte* et un *module aligneur*. L'*analyseur de contexte* est responsable de l'analyse des contextes des sources de données à aligner afin d'en extraire des informations sémantiques pertinentes qui serviront à enrichir les concepts composant ces sources de données. Comme expliqué plus loin, ce module contribuera également à la résolution du problème du mapping complexe en permettant de découvrir des combinaisons de concepts pertinentes pour le matching. Une fois "contextualisées", les sources de données sont alors traitées par le module aligneur. Ce module se caractérise par sa capacité de supporter plusieurs stratégies d'alignement

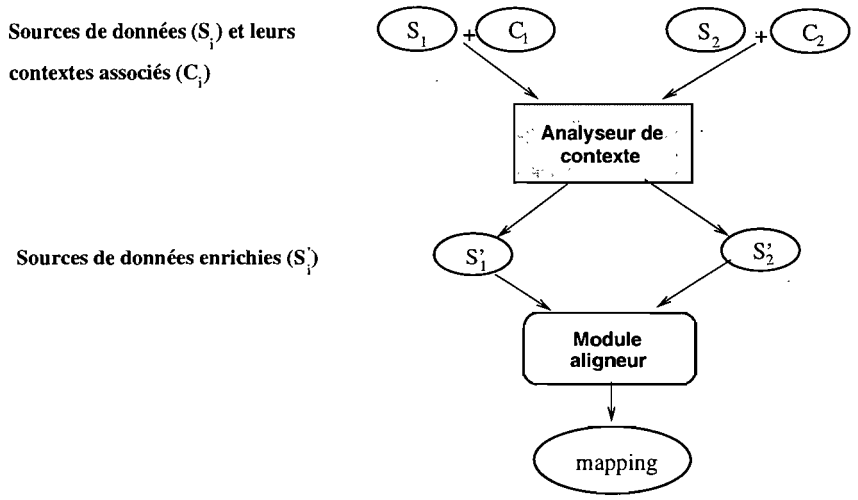


FIG. 4.1 – Processus global de matching d'INDIGO

qu'il peut sélectionner dynamiquement et exécuter en fonction des caractéristiques des concepts des sources de données à réconcilier. La figure 4.1 montre l'intervention de l'analyseur de contexte et du module aligneur dans le processus global de matching d'INDIGO.

4.2.1 Contextualisation des sources de données

Afin de remédier aux limitations (présentées dans la section 4.1) concernant la qualité et la quantité des ressources sémantiques, nous proposons d'étendre celles-ci en intégrant ce que nous avons appelé le *contexte externe d'une source de données.* En effet, nous pensons que le fait d'isoler une représentation de données de son contexte (en ne considérant que son schéma par exemple) réduit d'avance le lot d'informations cognitives pouvant être exploitées et qui se cache pourtant derrière la conceptualisation de cette représentation. En pratique, le contexte dans lequel s'inscrit une source de données constitue un gisement informationnel très précieux qui mérite d'être plus systématiquement exploré pour mieux cerner la sémantique des concepts.

Nous donnons, dans les paragraphes qui suivent, une définition de la notion de contexte d'une source de données et nous présentons une architecture générique de l'ana-

lyseur de contexte.

4.2.1.1 Définition de la notion de *contexte d'une source de données*

Le contexte d'une source de données est l'ensemble des informations sauvegardées sur un support informatique, appartenant à l'environnement externe de cette source et faisant partie de son domaine. Nous distinguons deux types de contexte :

1. Le *contexte descriptif* d'une source de données regroupe tous les fichiers de spécification décrivant les données ou leur environnement d'utilisation. Ces fichiers documentent les données selon différents niveaux d'abstraction. Leur contenu est généralement décrit sous forme de texte dans un format libre. Si la source est une base de données par exemple, le contexte descriptif pourrait être composé des documents suivants :
   - Cahier des charges exprimant les besoins des utilisateurs concernant les données et les traitements.
   - Dossier d'analyse et de conception.
   - Manuel utilisateur.
2. Le *contexte opérationnel* d'une source de données est composé des fichiers de gestion et manipulation des données. La description du contenu de ces fichiers respecte généralement une syntaxe formelle connue. Il peut s'agir entre autres de :
   - Programmes informatiques écrits dans différents langages.
   - Fichiers de requêtes de type SQL.

Pour chaque source de données, il importera de répertorier tous les documents pouvant composer ses contextes descriptif et opérationnel. L'analyse de ces documents (en plus des schémas et des données elles-mêmes) enrichira la base de connaissances qui contribuera à déduire le mapping le plus complet et le plus exact entre ces sources de données.

4.2.1.2 Analyse du contexte

L'objectif de l'analyse du contexte d'une source de données est d'enrichir cette dernière par des informations puisées à partir de ses contextes descriptif et opérationnel afin de

la préparer pour un alignement plus efficace.

FIG. 4.2 – Entrées et sorties d'un analyseur de contexte

En particulier, grâce à l'intégration du contexte il devient possible d'élaborer des mappings complexes mettant en correspondance des *combinaisons* de concepts sources et cibles (eg. (Rue, code_postal,ville) → Adresse_employé). Par exemple, si l'analyse du contexte permet de détecter, dans un programme informatique l'appel à la fonction **concat(rue, code_postal, ville)**, un nouveau concept *adresse_complète* peut être créé et ajouté à la source de données concernée et présenté comme un nouveau candidat pour le mapping (c.f. figure 4.3).

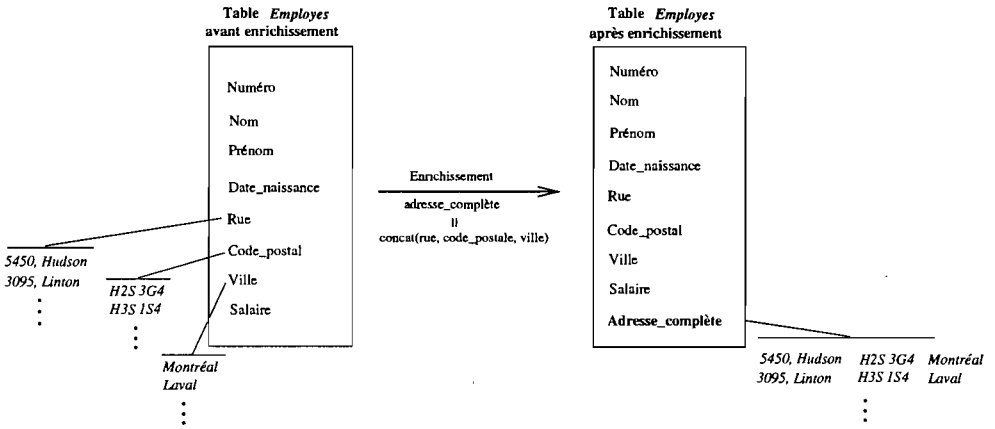


FIG. 4.3 – Exemple d'enrichissement d'une source de données par un concept complexe

Par ailleurs, l'analyse de contexte permet de mieux cerner la sémantique d'un concept de source de données ayant (1) un nom abrégé, tel que *n_fact* qui désigne le numéro d'une facture, ou (2) un nom générique comme *contact* indiquant le numéro de téléphone du service à la clientèle. En effet, ceci devient possible notamment en associant à ces concepts d'autres noms se trouvant fréquemment dans leur voisinage au niveau du contexte descriptif. Ce type d'enrichissement est particulièrement utile pour les stratégies de matching spécialisées dans la comparaison lexicale de noms de concepts.

L'enrichissement sémantique consistera donc (1) à associer à un nom de concept abbrégé ou générique un ensemble de noms explicites extraits du contexte permettant de mieux identifier sa sémantique ou (2) à ajouter à une source de données de nouveaux concepts complexes présentés comme de nouveaux candidats au mapping.

4.2.1.3 Architecture d'un analyseur de contexte

Le contexte d'une source de données se caractérise par une grande diversité de la nature des informations qui le composent : contenu formel (e.g. programme informatique ayant une syntaxe bien définie), semi-formel (e.g. tableau descriptif), ou non formel (e.g. description sous un format quelconque). Ainsi, pour assurer une analyse de contexte efficace nous proposons une approche d'analyse ciblant chaque type de contenu individuellement.

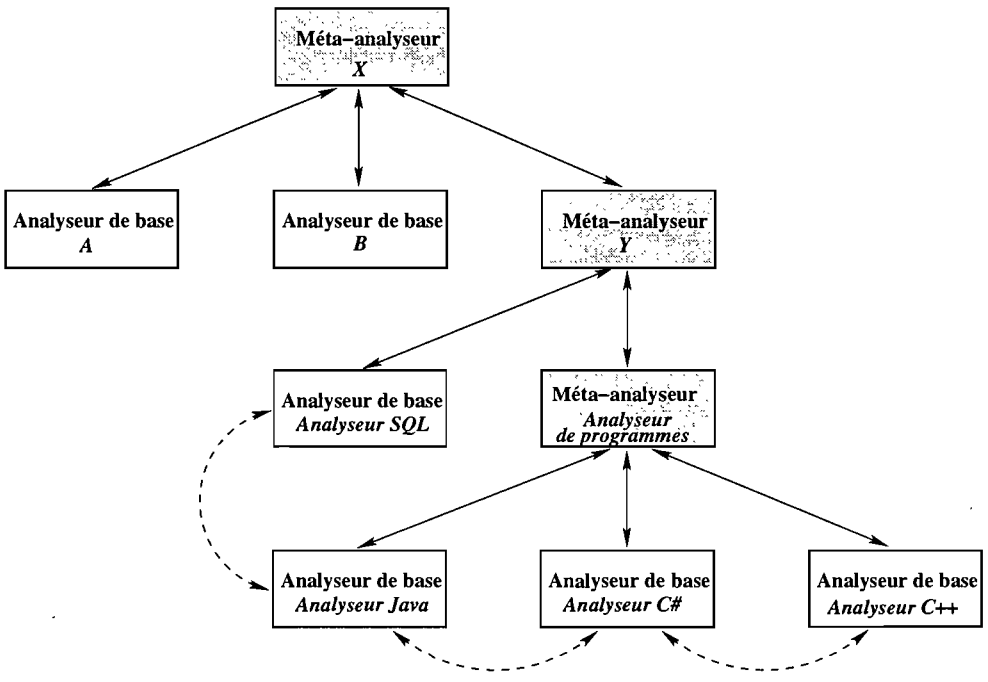


FIG. 4.4 – Architecture d'un analyseur de contexte

Nous préconisons une architecture basée sur plusieurs analyseurs de base supervisés par un méta-analyseur principal. Chaque analyseur de base est spécialisé dans le traite-

ment d'un type de contenu spécifique. Le rôle du méta-analyseur principal est de parcourir le contexte, de déterminer le type de chaque contenu et de faire appel à l'analyseur de base correspondant pour le traiter. Les résultats retournés par les analyseurs de base sont ensuite filtrés (élimination des redondances) et combinés par le méta-analyseur principal.

Par ailleurs, un analyseur de contexte possède une organisation hiérarchique (fig. 4.4). En effet, des analyseurs coordonnateurs occupant des noeuds dans la hiérarchie peuvent aussi jouer le rôle de méta-analyseurs en coordonnant d'autres analyseurs de base plus spécialisés. Par exemple, un analyseur de programme informatique supervise des analyseurs spécialisés chacun dans un langage de programmation particulier (Java, C++, C#, etc.).

Enfin, un analyseur de base peut directement collaborer avec un autre analyseur de base. Par exemple, un analyseur de programme peut faire appel aux services d'un analyseur SQL pour l'analyse d'une requête SQL rencontrée à l'intérieur d'un programme.

4.2.2 Processus de mapping

Le chapitre 3 montre que les systèmes basés sur des approches multi-stratégiques diffèrent dans le choix des stratégies qu'ils implémentent ainsi que dans la façon de les combiner. Chaque stratégie se base sur un type d'information sémantique particulier (schéma, instances, etc.) pour élaborer son mapping. Or, comme mentionné dans la section 4.1.2, ces systèmes restent des expériences isolées et sont liés à des types d'applications spécifiques. Afin de remédier à cette limitation, le module aligneur d'INDIGO utilise une architecture multi-stratégique facilitant les combinaisons dynamiques des stratégies de matching. Cette architecture lui confère la capacité d'adapter sa stratégie d'alignement (en sélectionnant et pondérant adéquatement les stratégies appliquées) en fonction des caractéristiques des sources de données à apparier et par conséquent d'optimiser sa performance globale de mapping.

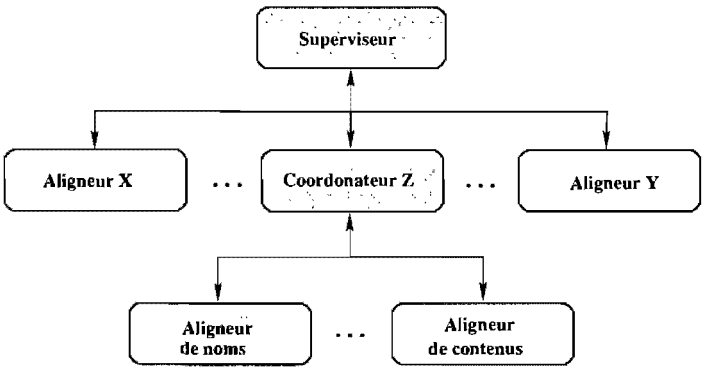


FIG. 4.5 – Architecture du module aligneur

4.2.2.1 Architecture

À l'instar de l'analyseur de contexte, nous préconisons pour le module aligneur une architecture hiérarchique (c.f. figure 4.5) composée de deux types de modules : *Aligneurs* et *Coordonnateurs*. Chaque aligneur implémente une stratégie d'alignement exploitant un ensemble de propriétés propres à une certaine catégorie de sources de données (e.g. nom de concepts, type de données, contenu textuel, etc.). Un coordonnateur se charge de la coordination des actions de deux ou plusieurs aligneurs et/ou coordonnateurs. Il a pour rôle essentiel de combiner les résultats de matching générés par ses modules subordonnés. Les aligneurs occupent les feuilles dans la hiérarchie de l'architecture tandis que les coordonnateurs sont représentés par des noeuds. Au sommet de la hiérarchie, un coordonnateur spécial appelé *superviseur* chapeaute l'ensemble des aligneurs et coordonnateurs du système. C'est au *superviseur* qu'incombe la responsabilité de l'élaboration du mapping final.

Notons que cette architecture remplit l'objectif de d'extensibilité que nous nous sommes fixé, puisque l'intégration d'un aligneur nécessitera simplement la modification du *coordonnateur* qui le supervisera. En plus, grâce à sa forme hiérarchique, elle se prête parfaitement à l'intégration d'un mécanisme de combinaison dynamique des stratégies. En effet, ce mécanisme peut être implémenté au niveau de chaque coordonnateur en choisissant de mettre sous sa responsabilité un ensemble homogène de stratégies de matching. Par exemple, il est possible de configurer l'architecture du module aligneur en

liant à chaque coordonnateur des stratégies d'alignement exploitant un même ensemble d'informations sémantiques. Le module aligneur pourra alors facilement coordonner et comparer les performances respectives des stratégies de matching opérant à partir des mêmes informations.

4.3 Conclusion

Notre approche INDIGO propose une solution globale d'alignement sémantique des données qui se distingue par sa flexibilité et son adaptabilité. En particulier, elle prend explicitement en compte le contexte des sources de données à aligner dans son processus global de mapping. Par ailleurs, elle s'appuie sur une architecture multi-stratégique d'alignement. Elle vise la réalisation des trois objectifs suivants :

1. Amélioration des stratégies de matching lexical en s'appuyant sur l'enrichissement des concepts à aligner par des noms issus du contexte descriptif. C'est ce que nous appelons la contextualisation du matching lexical et que nous détaillerons dans le chapitre 5.
2. Résolution du problème du mapping complexe en identifiant les combinaisons de concepts pertinentes au matching (à partir d'informations extraites du contexte opérationnel). L'alignement sémantique des concepts complexes est présenté au chapitre 6
3. Amélioration de l'adaptabilité par la mise en oeuvre d'une architecture multi-stratégique d'alignement qui permet la combinaison dynamique de stratégies de matching en fonction des caractéristiques des sources de données à aligner. Le mapping dynamique est présenté au chapitre 7 ;

La mise en oeuvre de ces trois aspects de notre approche d'alignement sémantique est réalisée par l'outil INDIGO que nous présenterons plus en détail au chapitre 8.

CHAPITRE 5

CONTEXTUALISATION DU MATCHING LEXICAL

La majorité des systèmes de mapping actuels se basent sur le matching lexical. Les techniques qui sont généralement utilisées pour implémenter ce type de matching procèdent (1) par comparaison de chaînes de caractères représentant des noms de concepts à aligner ou bien, (2) par rapprochement des synonymes associés à ces derniers en ayant recours à des dictionnaires externes. Malheureusement, ces solutions s'avèrent souvent inefficaces dans le cas où les sources de données à aligner sont très hétérogènes. En particulier, leur performance peut se dégrader considérablement lorsqu'elles se réfèrent à des dictionnaires génériques ne tenant pas compte des spécificités du domaine des sources de données à aligner. Pour surmonter ces limitations et mieux faire face au problème de l'hétérogénéité des données, nous proposons l'approche d'INDIGO qui tient compte du contexte pour élaborer le matching lexical. Elle se distingue par le fait de prévoir une étape préliminaire consistant à enrichir les sources de données avec des informations sémantiques extraites des documents composant leur contexte descriptif. Durant cette étape, chaque nom de concept est annoté par des informations sémantiques provenant de son domaine spécifique (i.e. son contexte). Une fois enrichies, les sources de données sont ensuite alignées par INDIGO qui tient alors compte de ces annotations.

Ce chapitre se compose de deux sections principales. La première section donne un aperçu des techniques de matching lexicales utilisées par les systèmes de mapping actuels et introduit l'approche d'INDIGO pour traiter ce type de matching. La deuxième section présente l'architecture utilisée par INDIGO pour supporter le matching lexical et explique son approche d'alignement.

5.1 Le matching lexical

Le matching lexical joue un rôle fondamental dans les applications spécialisées dans l'alignement d'ontologies [EBB+04, NM01, HCZ+06, JPE01]. Plus précisément, le proces-

sus de mapping tel qu'implémenté par cette catégorie d'applications est généralement exécuté en deux étapes : (1) les concepts sont tout d'abord réconciliés en utilisant une certaine technique de comparaison lexicale, ensuite (2) un algorithme de matching structurel est appliqué pour générer le mapping final. La phase du matching lexical a pour but d'élaborer les correspondances sémantiques entre les concepts en se basant sur la comparaison lexicale des noms les composant. Quant à l'algorithme d'alignement structurel, il génère un mapping en partant des résultats retournés par la première étape et en se basant sur l'hypothèse suivante : les concepts de deux sources de données sont susceptibles d'être similaires si les concepts situés dans leurs voisinages respectifs (e.g. frères et ancêtres dans des structures d'arbres) sont similaires aussi.

Au delà de l'alignement d'ontologies, les stratégies de matching lexical sont mises en oeuvre par nombre de solutions de mapping actuelles et méritent donc d'être mieux développées et améliorées. Les stratégies actuelles ont recours à deux catégories principales de techniques : (1) les métriques de comparaison des chaînes de caractères et (2) les fonctions d'évaluation de similarité lexicale. Les métriques de comparaison des chaînes de caractères font l'hypothèse que des concepts similaires sont susceptibles d'être modélisés par des noms similaires dans les différents schémas où ils se trouvent. Elles comparent les noms des concepts en rapprochant leurs morphologies lexicales respectives (e.g. nombre de caractères partagés en commun). La mesure de similarité de JARO et la distance d'édition [EBB+04] sont des exemples typiques de métriques de comparaison de chaînes de caractères. Quant aux fonctions d'évaluation de similarité lexicale, elles estiment la similarité entre les concepts en comparant leur sémantique. Ainsi, pour chaque concept une recherche est effectuée dans un dictionnaire externe donné (e.g. WordNet) afin d'en extraire l'ensemble des mots auxquels il est relié sémantiquement (e.g. par synonymie, hyponymie, etc.). Toutefois, en pratique, les techniques de matching lexical actuelles s'avèrent très souvent inefficaces lorsqu'il s'agit d'aligner des sources de données très hétérogènes. En effet, les métriques de comparaison de chaînes de caractères sont généralement incapables d'aligner correctement des noms abrégés ou encodés (e.g. $cl \rightarrow$ *command_line*). Par ailleurs, les fonctions d'évaluation de similarité lexicale sont rarement performantes puisque les dictionnaires génériques sur lesquels elles se basent ne

tiennent pas compte des spécificités du domaine des sources de données à aligner. Pour pallier ces limitations, notre approche INDIGO implémente une solution innovante basée sur l'exploration du contexte descriptif de chaque source de données à aligner. En effet, cette solution consiste à associer à chaque concepts des informations contextuelles émanant du domaine où ils ont été définis. Par exemple, certains concepts (e.g. *fournisseur*, *client* et *marché*) se retrouvant dans les voisinages du concept à aligner (e.g. *produit*) peuvent être utilisés pour enrichir sémantiquement ce dernier. Puisés à même les artefacts composant et documentant les sources de données (contexte descriptif), les informations d'enrichissement apportent des éléments supplémentaires permettant de préciser la sémantique de chaque concept qu'ils accompagnent et offrent ainsi un bon levier pour un matching lexical plus efficace.

5.1.1 Applications du matching lexical

La recherche dans le domaine du matching lexical est un chantier qui a largement été exploré ces 15 dernières années. Nous présentons ci-après un échantillon représentatif des deux principales catégories de techniques proposées dans ce domaine.

Concernant les métriques de calcul de similarité basées sur la comparaison de chaînes de caractères représentant les noms de concepts, nous énumérons ci-après cinq mesures parmi les plus populaires :

1. La distance de Levenstein [Lev66] : cette métrique compte le nombre minimal d'opérations d'insertions, de suppressions et de substitutions nécessaires pour transformer une chaîne de caractères vers une autre.
2. La distance de Needleman-Wunsch [NW70] : cette mesure est similaire à la distance de Levenstein sauf qu'elle attribue des poids plus élevés aux opérations d'insertion et de suppression.
3. La distance de Smith-Waterman [SW81] : l'algorithme de Smith-Waterman permet de trouver l'alignement optimal local entre 2 séquences d'éléments. Il permet de déterminer par exemple, les segments similaires composant respectivement deux chaînes d'ADN.

4. La similarité de Jaro-Winkler [Jar95, Win99] : l'évaluation de cette métrique se base sur le nombre et l'ordre des caractères en commun entre deux chaînes de caractères.
5. La similarité de Q-Gram [ST95] : cette mesure compte le nombre de q-grams (segments de longueur q) partagés entre deux chaînes de caractères.

Quant aux fonctions d'évaluation de similarité lexicale, elles ont été surtout développées dans le cadre d'applications de mapping telles que OLA [JEV05], ASCO [BLG04] et HCONE-merge [KKS04] pour servir les objectifs spécifiques de celles-ci. Tous ces systèmes ont recours à WordNet pour collecter des informations sémantiques supplémentaires (ensembles de mots) liés aux concepts à aligner. Par exemple, ASCO recherche les synonymes tandis que HCONE-merge retrouve les hyponymes. Typiquement, la similarité entre deux concepts est évaluée en calculant l'intersection entre les ensembles de mots collectés. Il est à noter que les expériences [JEV05] ont montré que les techniques basées sur la comparaison des chaînes de caractères sont généralement plus efficaces que les approches d'évaluation de similarité lexicale se référant à des dictionnaires en ligne. Parmi les nombreuses solutions de matching lexical citées en littérature, nous avons étudié plus en profondeur les systèmes Similarity Flooding (SF) [MGMR02], V-Doc [QHC06] et Cupid [JPE01]. Nous avons choisi ces trois applications pour les comparer expérimentalement avec l'approche de matching lexical implémentée par notre système INDIGO. Les tests expérimentaux, que nous détaillerons au chapitre 8, ont consisté à évaluer chacun de ces systèmes de mapping sur la base des deux études de cas suivantes :

- Alignement de deux schémas de bases de données obtenues à partir de deux applications open-source dans le domaine du commerce électronique : *Java Pet Store* [Mic05] et *eStore* [McU03].
- Alignement de deux sources de données réelles décrivant les programmes de cours donnés à l'Université de Cornell et à l'Université de Washington [ACC04].

Dans les deux cas, les artefacts des sources de données nous étaient accessibles. Nous les avons donc rassemblés pour composer le contexte descriptif des sources en vue de les analyser.

5.2 Architecture de Indigo

Rappelons que l'architecture de INDIGO (c.f. section 4.2) est composée de deux modules principaux : un analyseur de contexte et un module aligneur. L'analyseur de contexte prend les deux sources de données à aligner accompagnées des documents composant leurs contextes respectifs et procède à leur enrichissement avant de les passer au module aligneur pour leur mapping effectif. Dans les deux sections suivantes, nous allons discuter des rôles de ces deux modules dans le cadre d'un processus de matching lexical.

5.2.1 Rôle de l'analyseur de contexte

Le rôle de l'analyseur de contexte est d'enrichir une source de données par des informations sémantiques pertinentes extraites de son contexte descriptif afin de la préparer au matching lexical. Cet enrichissement consiste à associer à chaque concept des noms d'entités qui lui sont liées sémantiquement. Pour découvrir ces entités dans le contexte descriptif nous appliquons la règle heuristique suivante : "Deux concepts qui se retrouvent souvent à proximité l'un de l'autre sont susceptibles d'être liés sémantiquement"

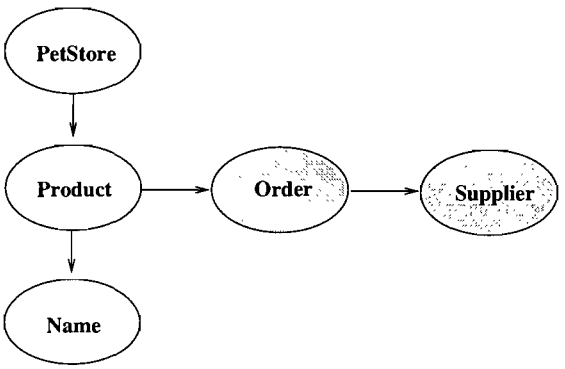


FIG. 5.1 – Graphe des dépendances entre les concepts de l'application *Pet Store*

Dans l'application *Pet Store* (notre première étude de cas), le modèle de base de données définit *name* comme un attribut de l'entité *product*. D'un autre côté, la documentation montre bien que les concepts *order* et *supplier* apparaissent très fréquemment dans le voisinage du concept *product*. En fait, de telles observations ont pu être réalisées

grâce à l'exécution des modules spécialisés de l'analyseur de contexte appliqués aux artefacts de l'application *Pet Store*. Pour chaque concept référencé par un chemin s dans une source de données XML, l'analyseur construit un graphe à partir des résultats d'analyse de ses modules. Ce graphe définit les dépendances sémantiques entre les concepts composant le chemin s et ceux provenant du contexte descriptif. Un tel graphe est appelé le *graphe de dépendances entre concepts* (**GDC**). Soit s le chemin XML d'un concept dans la source de données et soit $Concepts(s)$ l'ensemble des concepts composant le chemin s. Un **GDC** pour s est un graphe qui :

1. lie chaque concept $c \in Concepts(s)$ à son successeur c' dans s (tel que $c' \in Concepts(s)$). La position d'un concept dans la liste ainsi obtenue détermine un *niveau* d'enrichissement.
2. lie chaque concept $c \in Concepts(s)$ à une liste $elist(c)$ des concepts d'enrichissement extraits à partir du contexte de la source de données. La position d'un concept dans la liste d'enrichissement $elist(c)$ est appelée son *rang*.

Considérons par exemple le chemin *PetStore/product/name* dénotant le concept *name* dans le schéma XML de la base de données *PetStore*. La figure 5.1 montre le **GDC** établissant le lien de l'entité *product* avec son attribut *name* ainsi qu'avec la base de données *Pet Store* à laquelle il appartient. Ces concepts sont organisés verticalement. Le **GDC** montre aussi la relation de *product* avec deux concepts d'enrichissement, *order* et *supplier*, qui se retrouvent fréquemment dans son voisinage selon les conclusions de l'analyseur qui a analysé le contexte descriptif de la source de données *PetStore*. Ces concepts sont organisés horizontalement.

L'architecture de l'analyseur de contexte (c.f. figure 5.2) comprend deux modules principaux, chacun étant spécialisé dans l'extraction d'un type spécifique d'informations : le *collecteur de noms de concepts* et l'*extracteur des concepts complexes*. Le collecteur de noms de concepts est responsable de la collecte des concepts simples, extraits du contexte descriptif, en vue d'enrichir lexicalement une source de données. L'*extracteur de concepts complexes* se charge, quant à lui, de l'extraction des combinaisons de concepts à partir du contexte opérationnel. Ces combinaisons sont destinées à être ajoutées à une source

de données pour la préparer au mapping complexe. Dans les descriptions formelles, les combinaisons sémantiques de concepts sont plus facilement identifiables : entre autres, elles peuvent être repérées dans des formules, des déclarations de fonctions, etc. Ceci est l'une des raisons pour lesquelles INDIGO favorise l'exploration du contexte opérationnel (plutôt que le descriptif) pour l'extraction des concepts complexes. Dans ce chapitre, nous allons nous limiter à présenter le *collecteur de noms de concepts* uniquement. L'*extracteur des concepts complexes* sera discuté en détail dans le chapitre suivant.

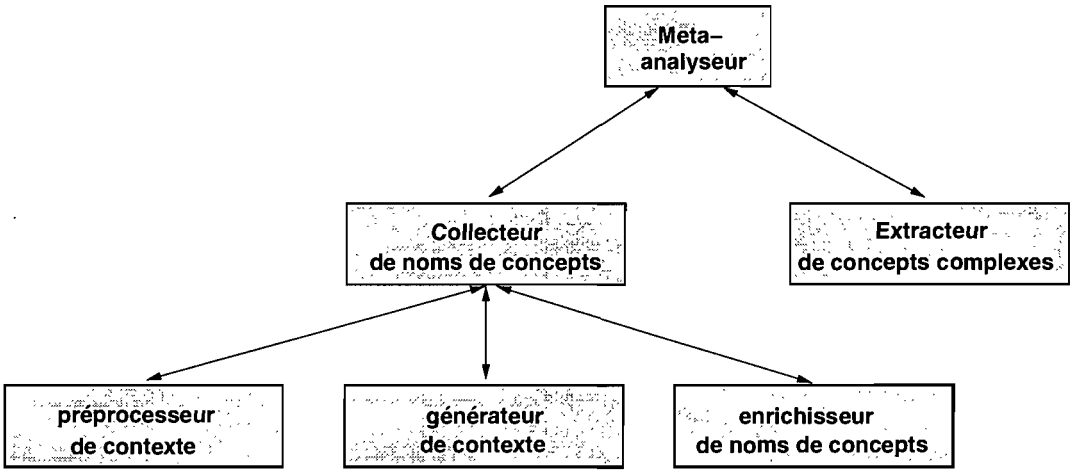


FIG. 5.2 – Architecture de l'analyseur de contexte

Tel illustré par la figure 5.2, les deux modules analyseurs que nous venons de présenter sont supervisés par un méta-analyseur qui coordonne leurs tâches respectives. Au niveau le plus bas, l'architecture offre deux modules utilitaires pour prétraiter et générer le contexte d'une source de données et finalement un troisième module pour enrichir cette dernière avec les informations ainsi extraites de son contexte.

5.2.1.1 Méta-analyseur

Le module méta-analyseur explore l'ensemble des documents composant le contexte informationnel de la source de données. Il classifie chaque document dans la catégorie descriptive ou bien opérationnelle en fonction de l'extension de son nom de fichier. Typiquement, les documents contenant du texte dans un format libre sont classés comme descrip-

tifs, alors que ceux présentant une structure interne composée de balises imbriquées sont plutôt considérés comme opérationnels. En fait, nous définissons deux listes disjointes d'extensions de noms de fichiers. La première liste concerne les documents descriptifs et contient ainsi des extensions de noms de fichiers telles que *.doc ou *.txt, tandis que la seconde regroupe des extensions de noms de fichiers opérationnels telles que *.java, *.cpp, ou *.sql. Pour classifier un fichier, le méta-analyseur contrôle simplement son extension et identifie la liste à laquelle il appartient. Les documents opérationnels sont explorés pour la découverte des concepts complexes. En effet, il est souvent plus facile de trouver des combinaisons de concepts dans les documents opérationnels où les regroupements de concepts sont plus systématiquement identifiables grâce à la présence d'opérateurs composant des formules (e.g. les formules arithmétiques sont facilement reconnaissables grâce aux opérateurs -, *, etc.). Quant aux artefacts descriptifs, ils sont plutôt analysés pour la collection de concepts simples pouvant être reliés sémantiquement aux éléments de la source de données. Les concepts d'enrichissement les plus pertinents se trouvent généralement dans les même phrases où un concept de la source de données est cité. Le méta-analyseur adresse chaque document en fonction de sa catégorie soit au collecteur de noms de concepts ou bien à l'extracteur de concepts complexes. Les résultats retournés individuellement par ces deux modules sont ensuite regroupés par le méta-analyseur qui construit une structure globale d'informations sémantiques destinée pour l'enrichissement de la source de données.

5.2.1.2 Collecteur de noms de concepts

Étant donné un ensemble de noms de concepts composant une source de données et un ensemble de documents descriptifs à analyser, le collecteur recherche chaque concept dans ces documents. Lorsqu'un concept est repéré, le collecteur rassemble tous les noms qui se retrouvent fréquemment dans son entourage pour finalement ne retenir que ceux qui sont les plus proches de lui, et ce, afin de les utiliser pour son enrichissement sémantique. Plus précisément, le collecteur procède à l'enrichissement de la source de données en deux temps : une phase de *prétraitement de contexte* est d'abord réalisée avant l'exécution de la procédure d'*enrichissement de source de données* en tant que telle.

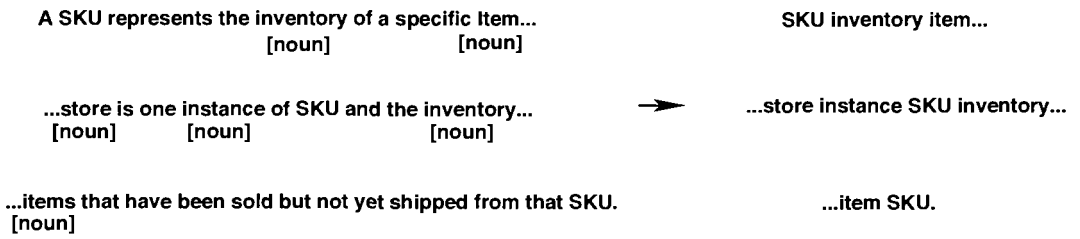


FIG. 5.3 – Prétraitement de contexte pour le concept SKU (génération de la partie correspondante du CC-Doc)

5.2.1.2.1 Prétraitement de contexte Le collecteur s'appuie sur deux modules internes appelés respectivement *Préprocesseur de contexte* et *Générateur de contexte* pour générer ce que nous appelons le *document de contextualisation de concepts* d'une source de données (en abrégé **CC-Doc**). Ce document est écrit en **XML**. Il est le résultat des fouilles et du filtrage effectués par le préprocesseur et le générateur de contexte. Le **CC-Doc** regroupe, pour chaque nom de concept de la source de données, l'ensemble des noms potentiellement pertinents qui entouraient ce concept dans l'un ou l'autre des documents composant le contexte de cette source de données. Ces noms sont stockés dans le **CC-Doc** sous une forme normalisée afin qu'un même concept n'ait qu'une seule représentation dans le **CC-Doc** bien qu'il puisse avoir été reconnu sous différentes formes (e.g. *inventory* et *inventories*), dans le contexte de la source de données. Pour générer le **CC-Doc**, nous utilisons le module *sentence splitter* et les outils d'annotation fournis par le cadre d'application **GATE** [lpg07]. L'API de WordNet est utilisée pour remplacer les noms par leurs formes de base.

Considérons par exemple l'application *Pet Store*. Sa base de données contient une table nommée *SKU* (i.e. Stock Keeping Unit). Le *Préprocesseur de contexte* analyse l'ensemble des documents descriptifs associés à l'application et rassemble toutes les phrases contenant le nom de concept *SKU*. Les phrases sont ensuite filtrées par le *Générateur de contexte* pour ne garder que les mots de type nom transformés en forme normale. L'ordre initial dans lequel un mot apparaît dans une phrase est bien entendu préservé. La partie droite de la figure 5.3 montre un extrait d'un **CC-Doc** après le traitement du concept *SKU*. Remarquons que le mot *items* de la troisième phrase a effectivement été

normalisé durant cette phase : le mot retenu dans le **CC-Doc** étant 'item'.

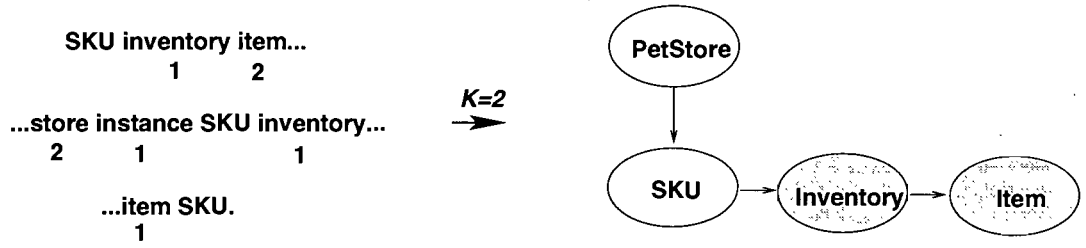


FIG. 5.4 – Enrichissement du concept SKU (génération de la partie correspondante du GDC)

5.2.1.2.2 Enrichissement de source de données Le **CC-Doc** élaboré lors du pré-traitement est utilisé par un troisième module du collecteur, appelé *Enrichisseur de noms de concepts*, permettant de générer un **GDC** pour chaque concept de la source de données. Considérons par exemple l'enrichissement du concept *SKU*.

Pour chaque phrase du **CC-Doc** lié au concept *SKU*, l'*enrichisseur de noms de concepts* marque chaque mot par une valeur représentant sa distance par rapport à *SKU*. Cette valeur indique le nombre de mots qui séparent le nom marqué du mot *SKU*, qu'il soit à droite ou à gauche. La partie gauche de la figure 5.4 montre le résultat d'une procédure de marquage. Une liste associant chaque nom marqué avec sa distance et sa fréquence d'apparition est ainsi construite. Les noms sont triés par ordre décroissant tout d'abord en fonction de leur fréquence d'apparition puis ensuite, en fonction de leur proximité par rapport au concept *SKU* (i.e. inversement à la moyenne des distances les séparant de *SKU*). Les meilleurs candidats sont retenus pour enrichir le concept *SKU*. Un seuil est utilisé pour limiter le nombre des concepts d'enrichissement à garder. Dans l'exemple précédent, étant donné un seuil K=2, les noms *item* et *inventory* sont sélectionnés pour enrichir le concept *SKU* et sont ainsi ajouté à son graphe **GDC**. Puisque le mot *inventory* est plus proche de *SKU* que le mot *item*, son rang dans le **GDC** précède celui de ce dernier (cf. partie droite de la figure 5.4).

Dans le cas particulier des noms composés, ceux-ci sont décomposés en sous-noms avant de procéder à l'enrichissement. La procédure de décomposition consiste à découper

un mot aux endroits pertinents i.e. au niveau des lettres majuscules ou de certains symboles spéciaux (e.g. shipAddress ou ship_address → (ship, address)). Chaque sous-nom est alors considéré comme un représentant du nom composé et est soumis à la même procédure d'enrichissement.

5.2.2 Rôle du module aligneur

Une fois enrichies par les informations nouvellement extraites de leur contexte, les sources de données sont soumises au module aligneur qui procède à leur mapping. L'architecture de ce module doit alors être configurée pour l'utilisation d'aligneurs supportant les stratégies adéquates de matching basées sur le contexte. Un processus d'alignement basé sur le contexte a pour objectif d'exploiter les informations d'enrichissement ajoutées aux sources de données afin d'améliorer sa performance globale.

Considérons un aligneur de noms, le matching basé sur le contexte est exécuté comme suit. Tout d'abord, un mapping est généré entre les deux sources de données à aligner sans tenir compte de leurs contextes. Ensuite, pour chaque paire de concepts un *facteur de similarité contextuelle* (**FSC**) est estimé en comparant leurs **GDC** (i.e. en tenant compte des concepts d'enrichissement ajoutés). Le **FSC** est une valeur normalisée entre 0 et 1, qui est utilisée pour pondérer l'augmentation de la similarité entre concepts en fonction de la similarité de leurs listes d'enrichissement respectives.

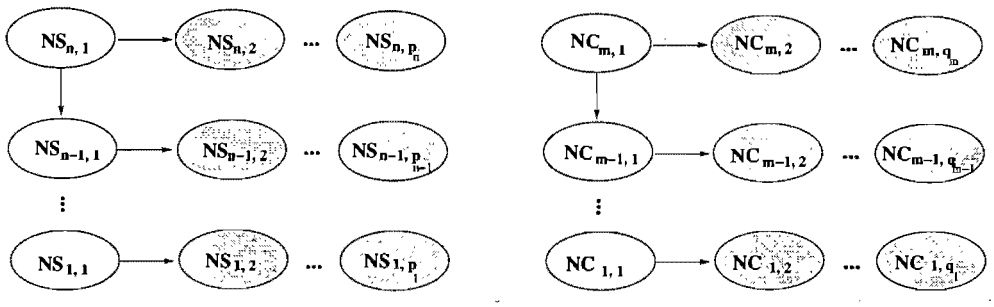


FIG. 5.5 – GDC d'un nom de concept source et d'un nom de concept cible

Pour montrer comment un **FSC** est calculé, considérons deux **GDC** associés respectivement à un concept source NS et un concept cible NC comme illustrés sur la figure 5.5.

NS_{ij} (resp. NC_{ij}) désigne le concept occupant le niveau i et le rang d'enrichissement j dans le **GDC** du concept source (resp. concept cible). Chaque ligne d'un **GDC** indique un niveau d'enrichissement décrit par une liste ordonnée de concepts d'enrichissement. Chaque colonne dénote donc le classement des concepts dans les différentes listes d'enrichissement. L'ordre des concepts dans une liste est établi en fonction de son degré estimé de pertinence contextuelle par rapport au contexte (cf. section 5.2.1.2.2). L'évaluation du **FSC** entre NS et NC est basée sur les deux observations suivantes :

- Les concepts situés à un niveau plus haut (dans un chemin XML ou dans un **GDC**) sont souvent ceux qui portent la sémantique d'un domaine et caractérisent mieux le contexte d'un concept. En revanche, les concepts situés au niveau le plus bas (tels que les attributs d'une table dans une base de données) véhiculent plutôt de l'information générique qui peut être indifféremment utilisée dans plusieurs contextes distincts. Par exemple, un attribut appelé *name* dans une source de données peut être utilisé pour qualifier plusieurs entités différentes (e.g. $Product \rightarrow name$, $Person \rightarrow name$, $NS_{2,1} \rightarrow NS_{1,1}$, etc.). En revanche, l'entité parent (e.g. *Product*, *Person* ou $SN_{2,1}$) est généralement le concept qui définit la sémantique contextuelle de son enfant. Pour cela, nous proposons de comparer les contextes (représentés par les **GDCs**) niveau par niveau et d'utiliser ensuite une méthode adéquate de pondération pour combiner les similarités obtenues pour chaque niveau.
- Considérons un ensemble de concepts qu'on peut comparer et classer en fonction de leur pertinence dans un contexte donné. Si l'on interprète ces mêmes concepts dans un contexte différent mais sémantiquement très similaire au premier, on peut raisonnablement espérer que le classement de ces concepts soit à peu près le même. Nous appuyant sur cette idée, nous proposons de comparer deux concepts NS et NC en évaluant la similarité des listes d'enrichissement de leurs **GDC** respectifs (niveau par niveau). Ainsi, il s'agira de comparer l'ordre des concepts de la ligne $NS_{1,1}...NS_{1,p_1}$ avec celui de la ligne $NC_{1,1}...NC_{1,q_1}$. La tâche n'est toutefois pas simple car les contextes ne sont décrits ni par les mêmes noms de concepts, ni par le même nombre de concepts. Par ailleurs, plusieurs concepts parasites peuvent avoir été introduits dans le **GDC** par le *collecteur de noms de concepts*. Aussi, la simila-

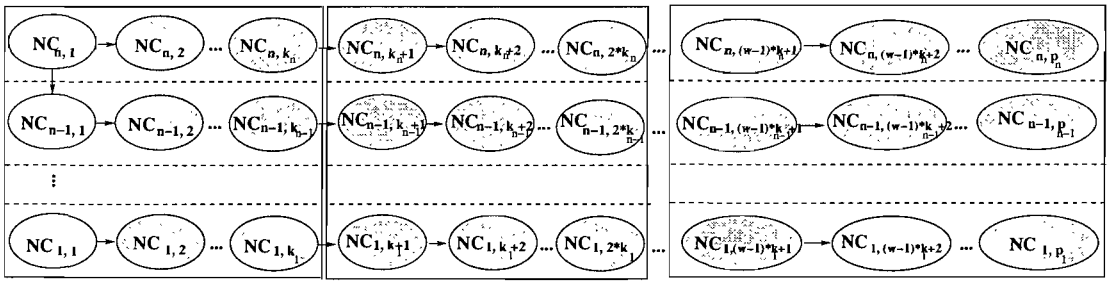


FIG. 5.6 – Subdivision du **GDC** d'un concept

rité entre des chaînes de classement de concepts peut être difficile à évaluer, si elles sont de tailles très différentes. En effet, le *collecteur* n'impose pas de barre inférieure sur le nombre de concepts d'enrichissement à chaque niveau. Afin d'atténuer les effets de bruit et faciliter la comparaison des listes d'enrichissement, nous subdivisons chaque liste en un certain nombre prédéfini de sous-ensembles de concepts de même cardinalité. Pour comparer deux listes d'enrichissement, il suffit alors de comparer leurs sous-ensembles de concepts deux à deux. En fait, cette comparaison est basée sur l'évaluation de l'intersection de sous-ensembles (ce qui aide à masquer le bruit) au lieu qu'elle le soit sur l'égalité de concepts.

La figure 5.6 montre la subdivision, en w boîtes, des concepts d'enrichissement dans un **GDC**. Sur un niveau donné, les boîtes doivent être presque de même taille i.e. doivent contenir à peu près le même nombre de concepts d'enrichissement. Supposons qu'au niveau i il y a p_i concepts et que $p_i = w * k_i + r_i$, les r_i premières boîtes contiendront donc $k_i + 1 = \lceil p_i/w \rceil$ concepts alors que les $w - r_i$ boîtes suivantes en contiendront k_i.

Soit B_i, avec $i \in \{1, ..., w\}$, la i^{eme} boîte dans un **GDC**. On note $B_{i,j}$ la sous-liste de concepts d'enrichissement du niveau j dans la boîte B_i. Notons que les **GDCs** respectifs des concepts source et cible (tels qu'illustrés par la figure 5.5) sont subdivisés tous les deux en un même nombre de boîtes w. Ainsi, la similarité entre les boîtes B_i et $B'_{i'}$, où $i, i' \in \{1, ..., w\}$, est donnée par la formule suivante :

$$BoxSim(B_i, B'_{i'}) = \frac{\sum_{x=0}^{\min(n,m)-1} \lambda^{\min(n,m)-x-1} Inter(B_{i,n-x}, B'_{i',m-x})}{\sum_{x=0}^{\min(n,m)-1} \lambda^{\min(n,m)-x-1}}$$

$Inter(B_{i,n-x}, B'_{i',m-x})$ est une fonction retournant 0 si $B_{i,n-x} \cap B'_{i',m-x}$ est vide, et

$\gamma \leftarrow 0$
$sim \leftarrow 0$
$while\ \gamma\ <\ w\ and\ sim = 0\ do\ \{$
$\quad sim\ \leftarrow\ BoxSim(B(1), B'(\gamma+1))$
$\quad \gamma\ \leftarrow\ \gamma+1$
$\}$
$return\ (1-\frac{\gamma}{w})BoxSim(B(1), B'(\gamma))$

FIG. 5.7 – Pseudo-code pour le calcul du FSC de deux GDCs composés respectivement des deux ensembles de boîtes B et B'

retournant 1 sinon. λ est une constante positive, supérieure à un, qui est déterminée de façon empirique.

Ainsi, nous proposons de calculer le facteur de similarité contextuelle ($FSC(B, B')$) (cf. le pseudo-code illustré par la figure 5.7) comparant les GDCs de deux concepts (représentés par les ensembles de boîtes B et B' respectivement) en utilisant la formule suivante :

$$FSC(B, B') = (1-\frac{\gamma}{w})BoxSim(B_1, B'_\gamma)$$
où γ est l'indice le plus petit tel que $BoxSim(B_1, B'_\gamma) \neq 0$.

Il est à noter que l'expérience [BV07] nous a montré que plus les concepts d'enrichissement se situent loin de la tête d'un GDC, plus ils deviennent bruyants. Par conséquent, nous avons choisi de considérer uniquement la comparaison entre la boîte tête d'un GDC source et la première boîte similaire d'un GDC cible (i.e. B_1 et B'_γ) et inversement. Plus précisément, l'évaluation du facteur de similarité contextuelle global (**FSCG**) est alors donnée par la formule suivante :

$$FSCG = \frac{FSC(B,B')+FSC(B',B)}{2}$$

La figure 5.8 illustre un mapping partiel entre les sources de données des applications *Pet Store* et *eStore*, avant et après le processus d'enrichissement. Dans cet exemple, le concept *unitprice* (désignant le prix mentionné sur une ligne de facture d'un article) ne devrait pas être aligné avec *unit_pr* (représentant le prix d'un article en stock). Avant toute tentative d'enrichissement (c.f. partie gauche de la figure 5.8), les évaluations de

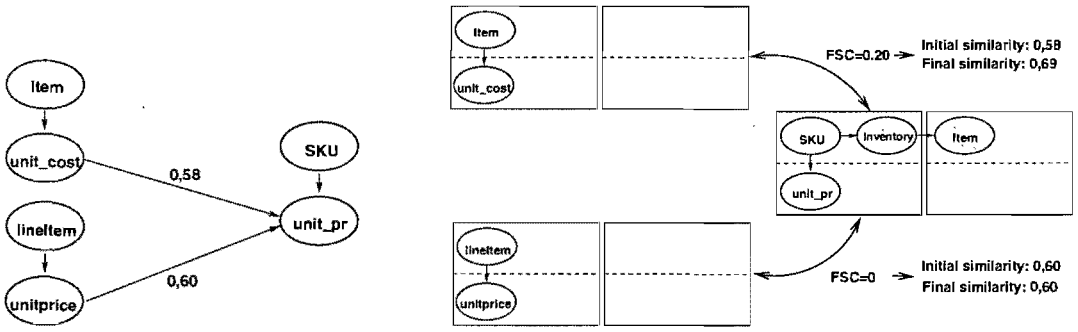


FIG. 5.8 – Alignement avant et après le processus d'enrichissement

similarités ont malheureusement favorisé ce dernier matching au lieu de la paire attendue (*unit_cost*, *unit_pr*). Cette situation était bien entendu prévisible puisque les évaluations de similarité étaient uniquement basées sur la métrique de similarité lexicale Jaro. Lorsque le processus d'enrichissement de noms est exécuté, les similarités sont évaluées différemment : la similarité entre deux concepts est estimée en tenant compte de la similarité de leur voisinage. Comme illustré par la partie droite de la figure 5.8, le concept *SKU* a été enrichi par deux noms de concepts *inventory* et *item*. En fixant le paramètre w de subdivision du **GDC** à 2, nous constatons que la seconde boîte du **GDC** de *unit_pr* partage le concept *item* avec la première boîte du celui de *unit_cost* à son premier niveau. En revanche, les boîtes du **GDC** du concept *unit_pr* ne partagent aucun concept avec celles du **GDC** de *unitprice*. Si l'on suppose que la constante λ est fixée à 4 dans la formule de *BoxSim*, alors le *facteur de similarité contextuelle* entre les concepts *unit_cost* et *unit_pr* sera estimé à 0,20 tandis que celui entre *unit_cost* et *unitprice* sera évalué à 0. Par conséquent, comme montré par la figure 5.8, la paire des concepts *unit_cost* et *unit_pr* voit la valeur de sa similarité augmenter de 20% pour atteindre un résultat final de 0,69. Pendant ce temps, la similarité entre *unitprice* et *unit_pr* reste inchangée. Dans ce simple exemple, on constate que l'enrichissement de sources de données a effectivement contribué à l'amélioration de la qualité du mapping. Comme nous le verrons dans nos études de cas (chapitre 8), cette amélioration est également remarquable à plus large échelle.

5.3 Conclusion

Ce chapitre présente une solution innovatrice proposée par INDIGO pour le matching lexical. INDIGO s'appuie sur une architecture composée de deux modules principaux : un analyseur de contexte et un module aligneur. L'analyseur de contexte enrichit les sources de données par des informations sémantiques extraites des artefacts appartenant à leurs contextes descriptifs respectifs avant de les délivrer au module aligneur qui procède à leur mapping. Nous pensons que le succès d'une telle approche peut ouvrir la voie à de nouvelles solutions adaptatives basées sur le contexte, capables de faire face à la diversité des données sur le Web et à la réconciliation de leur sémantique. En effet, les occasions de profiter des informations contextuelles sont nombreuses puisqu'une grande partie des sources de données sur le Web sont documentées. Par exemple, un catalogue de vente de produits «en ligne» dépend souvent d'une application de base de données documentée dans le back-office et est par conséquent inévitablement liée à un ensemble d'artefacts.

CHAPITRE 6

ALIGNEMENT SÉMANTIQUE DE CONCEPTS COMPLEXES

Le matching complexe (plusieurs à plusieurs) soulève un problème beaucoup plus difficile que celui du matching simple (un à un). Ceci est dû au fait que l'espace de recherche des combinaisons de concepts d'une source de données peut être très grand. Ce chapitre présente notre approche telle qu'implémentée par INDIGO pour résoudre ce problème. L'analyse de contexte permet à INDIGO d'identifier un certain nombre de concepts complexes qu'il ajoute alors aux sources de données à aligner. Il procède ensuite au mapping des sources de données, opération dès lors facilitée par l'enrichissement sémantique qui vient d'être réalisé.

Le reste du chapitre est organisé comme suit. La section 6.1 donne un aperçu sur l'approche du matching complexe d'INDIGO. La section 6.2 discute des récents travaux sur le matching sémantique complexe. Enfin, La section 6.3 décrit l'implémentation actuelle de l'*analyseur de contexte* et du module aligneur d'INDIGO.

6.1 Le matching complexe

À ce jour, la majorité des solutions actuelles se sont contentées de traiter le matching simple (e.g. *postal_code* → *zip_code*) même si le matching complexe (e.g. *concat(street,city)* → *address*) est souvent nécessaire en pratique. Cette réalité peut s'expliquer par le fait que la mise en correspondance sémantique entre concepts complexes soulève un problème beaucoup plus difficile que celui de l'alignement simple. En effet, alors que l'espace de recherche est fini dans le cas de l'alignement simple[1], il peut être infini dans le cas d'un alignement complexe. En effet, il ne s'agit plus simplement de faire le matching d'un concept avec un autre, il faut également considérer la façon dont certains concepts seront combinés pour être alignés avec un autre. Or, non seulement doit-on maintenant considérer le nombre de combinaisons possibles ($\sum_{i=1}^{n} \frac{n!}{(n-i)!i!}$ pour une source de taille

[1]Il s'agit du produit cartésien des ensembles finis de concepts sources et cibles

n) mais on doit également tenir compte du nombre possiblement important d'opérateurs (*concat*, $+$, $*$, ...) utilisés pour combiner ces concepts (jusqu'à $n\sum_{i=1}^{n}\frac{n!}{(n-i)!i!}$ combinaisons de concepts s'il y a n combinateurs possibles). Ceci peut éventuellement réduire de façon dramatique les performances du matching. Pour faire à face à ce problème difficile, nous proposons une solution qui permet de découvrir les combinaisons de concepts pertinentes et de les aligner sans pour autant devoir parcourir tout l'espace de recherche des candidats potentiels. Pour ce faire, INDIGO innove en exploitant de nouveau le contexte des sources de données.

6.1.1 Choix du type de contexte

Rappelons qu'INDIGO classe en deux ensembles principaux les documents composant le contexte informationnel d'une source de données. Le premier ensemble, appelé le contexte descriptif, regroupe tous les fichiers disponibles, documentant ou spécifiant une source de données, qui sont produits au cours du développement (e.g. document de spécification des besoins). Il s'agit de documents écrits généralement dans un style informel ou semi-formel. Quant au second ensemble, on l'appelle le contexte opérationnel. Il est composé d'artefacts formels tels que les programmes, les formulaires ou les fichiers XML. INDIGO fait l'exploration du contexte opérationnel pour faire l'extraction des concepts complexes.

6.1.2 Extraction des concepts complexes

Chaque concept complexe extrait du contexte opérationnel, est représenté dans INDIGO par un triplet <*nom*, *TypeDeDonnées*, *CombinaisonDeConcepts*> dont les compoants dénotent respectivement le nom du concept complexe, son type de données et la combinaison de concepts à laquelle il est associé (e.g. <*totalprice*, *float*, *price*(1+taxes)*>). Pour découvrir efficacement les concepts complexes, INDIGO s'appuie sur un ensemble de modules analyseurs de contexte respectivement spécialisés dans le traitement d'une catégorie particulière d'artéfacts (e.g. programmes, formulaires, etc.). Chaque analyseur applique un ensemble de règles d'extraction heuristiques pour extraire les concepts com-

plexes. Une fois extraits, les concepts complexes sont ajoutés aux sources de données comme de nouveaux candidats pour la phase du matching. Comme dans le cas du matching simple, cette opération est appelée *enrichissement* de la source de données.

Notons qu'INDIGO a été utilisé pour l'alignement sémantique de quatre schémas de bases de données provenant respectivement de quatre applications développées pour le commerce électronique. En plus des applications *Java Pet Store* [Mic05] et *eStore* [McU03] introduites dans le chapitre précédent, nous avons utilisé deux autres applications : *PetMarket* [Ado07] et *PetShopDNG* [Dot03]. Le code source de chacune de ces applications étant disponibles en libre accès sur Internet, nous avons profité de l'occasion pour les utiliser pour nos expériences. Les résultats et conclusions de nos expérimentations seront détaillées au chapitre 8.

6.2 Applications du matching complexe

Au meilleur de notre connaissance, les seuls travaux ayant directement abordé le matching complexe sont ceux décrits dans [XE03], [He04] et [RLD+04]. Nous présentons ci-après un aperçu de ces différentes approches.

Dans [XE03], les auteurs préconisent une approche qui aligne les sources de données avec une ontologie intermédiaire du même domaine, construite manuellement. L'ontologie joue le rôle de passerelle facilitant la découverte des matchings complexes. En particulier, elle définit des concepts complexes génériques et leurs relations (e.g. agrégation, généralisation ou spécialisation) avec d'autres concepts. Cette approche s'avère globalement efficace mais présente l'inconvénient d'exiger le développement manuel d'une ontologie pour chaque domaine spécifique.

Dans [He04], les auteurs décrivent une application cadre appelée **DCM** qui permet de découvrir des combinaisons sémantiques de concepts à l'intérieur de requêtes, destinées aux bases de données, qui sont émises sur Internet. **DCM** exploite les patrons de co-occurrence de concepts au sein de ces requêtes. Même si **DCM** parvient à des résultats de bonne précision, l'application n'est pas toujours très performante dans son traitement du matching complexe. En effet, plusieurs combinaisons de concepts (e.g. $total = (1 +$

taxeRate) $*$ *unitPrice*) ont peu de chance d'être identifiées si l'on se base uniquement sur les co-occurrences de concepts présentes dans les interfaces de requêtes.

Mentionnons enfin le travail de Dhamankar et al [RLD+04] où l'on présente le système **iMAP** qui découvre des concepts complexes en fouillant dans l'espace de toutes les combinaisons possibles de concepts. Ce système ayant déjà été présenté dans le chapitre 3, nous ne rappellerons ici que quelques unes des fonctionalités et caractéristiques qui nous intéressent. Pour assurer l'efficacité de sa fonction de recherche, **iMAP** emploie plusieurs modules de recherche, chacun étant spécialisé dans l'exploration d'un sous-espace composé d'un seul type de combinaisons (e.g. sous-espaces des combinaisons textuelles ou arithmétiques). En plus, **iMAP** utilise une technique itérative appelée *Beam Search* pour contrôler et borner la recherche des combinaisons à aligner. Même s'il implémente une approche intéressante, le système **iMAP** présente quelques inconvénients importants. Entre autres, étant donné que la technique du *Beam Search* limite le nombre maximal d'itérations de calcul, il est possible que certaines combinaisons pertinentes au matching ne soient pas découvertes.

Dans cet état de l'art, notre système INDIGO s'inscrit parmi les outils de matching simple et complexe à stratégies multiples. Il se distingue des autres en offrant une architecture hiérarchique flexible facilement configurable et adaptable aux besoins de l'alignement. Il présente surtout la particularité de tenir compte du contexte des sources de données au cours du processus d'alignement. Le système INDIGO est ainsi capable de générer efficacement aussi bien les matchings simples que complexes sans avoir recours à une ontologie intermédiaire décrivant le domaine. De ce point de vue, INDIGO surmonte beaucoup de limitations des systèmes d'alignement existants. En contre partie, INDIGO est inévitablement dépendant de la disponibilité, de la qualité et de la pertinence des artéfacts composant le contexte des sources de données à aligner. On peut toutefois raisonnablement espérer la disponibilité de telles ressources en pratique, considérant que l'utilisation d'une source de données implique habituellement l'existence de documents de sa décription et/ou d'un ensemble d'outils (e.g. programmes, requêtes SQL, etc.) de sa gestion.

6.3 Particularités de l'architecture d'Indigo pour le matching complexe

Rappelons que pour prendre en charge l'analyse du contexte et l'alignememt sémantique, INDIGO présente une architecture composée de deux modules principaux : un *analyseur de contexte* et un module *aligneur* (cf. figure 4.1). Le module *analyseur de contexte* reçoit les sources de données à aligner, ainsi que les documents composant leur contexte respectif, et procède à leur enrichissement avant de les livrer au module *aligneur* qui effectuera le matching. Les deux sections suivantes présentent les rôles respectifs de ces deux modules dans le cadre d'un processus de matching complexe.

6.3.1 Rôle de l'analyseur de contexte dans le cadre du matching complexe

L'*analyseur de contexte* comprend lui-même deux modules principaux, chacun d'eux étant spécialisé dans un type spécifique d'extraction et d'enrichissement de concepts. 1) Le *collecteur de noms de concepts* (cf. chapitre 5) explore le contexte descriptif d'une source de données pour identifier les noms (simples) de concepts qui peuvent être liés, par voisinage dans une phrase par exemple, à ceux se trouvant dans le schéma de la source de données. 2) L'*extracteur de concepts complexes* analyse le contexte opérationnel pour extraire les combinaisons de concepts (i.e. les concepts complexes). Dans les deux cas, les concepts nouvellement extraits sont intégrés aux sources de données respectives pour enrichir la sémantique des concepts présents dans leurs schémas. Ces deux modules analyseurs sont supervisés par un méta-analyseur principal qui coordonne leurs actions et voit à l'enrichissement des sources de données. La figure 6.1 a) montre l'architecture actuelle de notre *analyseur de contexte*. Tel qu'illustré, le module extracteur de concepts complexes fait appel à des analyseurs de base spécialisés. Ceux-ci sont représentés par des boîtes blanches. À ce jour, des analyseurs spécialisés ont été conçus pour l'analyse de formulaires, de programmes et de requêtes SQL.

Chaque module analyseur implémente un certain ensemble de règles heuristiques. Ces règles peuvent facilement être adaptées ou étendues pour répondre aux besoins spécifiques liés aux documents à analyser. Tous les modules composant l'*analyseur de contexte* sont donc extensibles et peuvent être adaptés pour supporter les différents

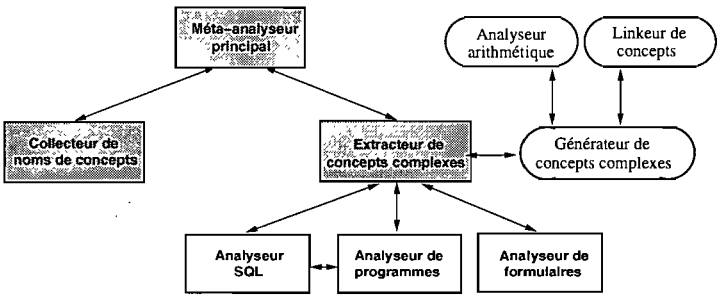


FIG. 6.1 – L'*analyseur de contexte* d'INDIGO

types d'artefacts présents dans le contexte des sources de données. Par exemple, il est envisagé de développer un nouveau module pour l'analyse des opérations présentes dans les sources de données objets. Cet analyseur pourra réutiliser certaines des heuristiques mises en oeuvre par l'analyseur de programmes pour identifier de nouveaux concepts complexes.

Les prochains paragraphes proposent une explication plus détaillée du mode d'utilisation des règles d'extraction heuristiques ainsi que de l'implémentation des *analyseurs de base* et des *méta-analyseurs* impliqués dans le processus du matching complexe.

6.3.1.1 Analyseurs de base

Les analyseurs de base sont responsables de l'exploration du contexte opérationnel. Ils se basent sur des règles heuristiques pour l'extraction des concepts complexes. Une règle d'extraction a la forme suivante :

$$nomRegle :: PS_1||PS_2...||PS_n \rightarrow actionExtraction$$

La partie gauche définit une disjonction de patrons syntaxiques (notés PS) que les analyseurs de base tentent d'identifier au sein d'un document au cours de son analyse. Un PS est une expression régulière qui peut contenir les variables *name*, *type*, exp_1, exp_2, ... exp_n. Lorsqu'un analyseur de base reconnaît l'un des patrons syntaxiques définis dans la partie gauche de la règle d'extraction, les variables se voient affecter une valeur par pattern matching. L'action décrite dans la partie droite de la règle est alors exécutée. Cette

action a pour but de construire un concept complexe $< nom, type, combinaison\_de\_concepts >$ à partir des valeurs affectées aux variables de la partie gauche. Considérons par exemple l'heuristique selon laquelle une méthode d'accès dans une classe est fortement susceptible de contenir une combinaison pertinente de concepts à l'intérieur de sa clause *return*. Cette heuristique est implémentée par notre *analyseur de programmes* par la définition d'une règle :

- dont la partie gauche contient un ensemble de patrons PS_i $(1 \leq i \leq n)$ reconnaissant les méthodes d'accès (e.g. dans Java : $PS_1 = \{$`public` *type* `get`*nom* * `return` $exp_1\}$, ou dans C# : $PS_2 = \{$`public` *type nom* { `get` * `return` $exp_1\}$).
- et dont la partie droite est une fonction retournant un concept complexe $<$*nom*, *type*, $exp_1>$ construit à partir des variables instanciées du patron PS_i identifié.

```
public string fullName {
        get {
                if (firstName.Length>0)
                        return firstName + ' ' + lastName;
                else
                        return lastName;
        }
}
```

```
public double getTotalCost() {
        return quantity * unitCost;
}
```

FIG. 6.2 – (À gauche) Partie d'un programme C# appartenant au contexte de *eStore*. (À droite) Partie d'un programme java appartenant au contexte de *PetStore*

Chaque analyseur de base applique son propre ensemble de règles d'extraction heuristiques sur chacun des artéfacts qui lui ont été assignés. La figure 6.2 montre deux extraits de programmes, l'un écrit en java et l'autre en C#. La règle heuristique susprésentée, appliquée à ces extraits, permet de générer les deux concepts complexes suivants : $<$*fullname*, *string*, $firstname+' '+lastname>$ et $<$*TotalCost*, *double*, $quantity * unitCost>$. Dans les paragraphes qui suivent, nous donnons un aperçu des règles heuristiques implémentées par chacun des analyseurs de base utilisés dans l'architecture actuelle d'INDIGO.

Analyseur de programmes. Un analyseur de programmes est responsable de l'analyse des fichiers contenant du code source. En plus de la règle concernant les méthodes d'accès et les mutateurs, cet analyseur implémente une autre heuristique portant sur les constructeurs. Nous considérons que le rôle d'un constructeur, au

sein d'une classe C, vise généralement la construction d'un objet composite du type C (e.g. *creditCard*) à partir des éléments reçus en paramètres (e.g. *cardNumber, expiryDate, cardType*). Cette heuristique s'appuie donc sur le fait que l'agrégation de ces paramètres forme vraissemblablement un concept complexe correspondant au concept représenté par C.

Analyseur de formulaires. Un *analyseur de formulaires* est un module spécialisé dans la découverte des concepts complexes constitués à partir des informations présentes dans les formulaires. Sa stratégie se base sur l'exploitation de la sémantique communément associée aux tables se trouvant dans ces formulaires. Dans le cas de l'application *eStore* par exemple, on trouve un formulaire contenant une structure tabulaire composée de deux colonnes. La première colonne contient le mot *Street* tandis que la seconde est composée de deux champs de saisie, appelés respectivement *Address1Label* et *Address2Label*. L'analyseur de formulaires d'INDIGO permet la reconnaissance de ce patron tabulaire typique. Selon l'heuristique actuellement implémentée, le concept figurant dans la première colonne est interprété comme un concept composite dont la valeur correspond à la concaténation des deux libellés de la seconde colonne.

Analyseur SQL. L'analyseur SQL est spécialisé dans l'analyse de requêtes SQL. Ses règles d'extraction ciblent les clauses SELECT contenant des expressions pouvant représenter des combinaisons de concepts, comme c'est le cas de *concat* dans l'exemple suivant : `SELECT street, concat(address_line_1, address_line_2), FROM Person`. Si aucune combinaison n'a pu être identifiée à l'intérieur de la requête SELECT, un nouveau concept complexe est néanmoins généré par simple concaténation de tous les concepts composant cette requête.

6.3.1.2 Méta-analyseurs.

Chaque méta-analyseur est spécialisé dans le traitement d'un ensemble d'artéfacts composant le contexte informationnel. Son rôle consiste essentiellement à classifier ces artéfacts et à assigner chacun d'entre eux au module fils approprié. Pour ce faire, un méta-

analyseur s'appuie sur des heuristiques exploitant les extensions de noms de fichiers ou bien analysant la structure interne des fichiers. Les paragraphes suivants donnent une description sommaire des méta-analyseurs implémentés dans INDIGO. À noter que le *collecteur de noms de concepts* n'est pas présenté ici car il n'est pas directement impliqué dans le matching complexe et car il a déjà été introduit au chapitre 5 dans le cadre du matching simple.

6.3.1.2.1 Méta-analyseur principal. Le méta-analyseur principal se trouve à la tête de l'architecture de l'*analyseur de contexte*. Il est chargé de la coordination du *collecteur de nom de concepts* et de l'*extracteur de concepts complexes*. Il enrichit les sources de données en y intégrant les concepts simples et complexes fournis respectivement par ces deux modules. Pour le cas des concepts complexes, l'étape d'enrichissement nécessite non seulement l'intégration des noms dans la source de données mais également des valeurs associées aux combinaisons de concepts[2]. Ces valeurs sont calculées en exécutant une requête SQL sur la base de données.

6.3.1.2.2 Extracteur de concepts complexes. L'*extracteur de concepts complexes* se charge de la coordination des actions des modules spécialisés dans l'extraction des concepts complexes appliqués actuellement aux programmes, formulaires et requêtes SQL. En plus, il s'appuie sur un module interne, appelé *générateur de concepts complexes* (c.f. figure 6.1), responsable d'assurer la cohérence de l'enrichissement des sources de données en exécutant les deux tâches suivantes :

- *Validation et harmonisation :*
  - Dans le cas où le concept complexe est de type numérique, un analyseur arithmétique est utilisé pour vérifier la validité de l'expression qu'il dénote. Si un problème est rencontré pendant l'analyse, le concept est automatiquement rejeté.
  - En ce qui concerne les concepts complexes s'exprimant comme une chaîne de caractères, la combinaison de concepts qu'ils dénotent se présente généralement sous forme d'une liste de concepts séparés par des virgules ou par des opérateurs

[2]En effet, les concepts partageant des valeurs similaires sont susceptibles d'être similaires.

'+'. Chaque concept est analysé pour savoir s'il s'agit d'un littéral ou bien d'une constante définie dans un programme. Une expression de concaténation est ensuite créée à partir de ces concepts en respectant le format de la source de données à enrichir.

- *Établissement de liens entre concepts :* Les noms inclus dans les expressions des concepts complexes doivent être remplacés par leurs correspondants dans les sources de données. Cette mise en correspondance est établie grâce à l'utilisation de la métrique de similarité *JaroWinkler*. Un seuil de similarité est arbitrairement fixé à une valeur élevée (e.g. fixé à 0.85) pour ne retenir que les combinaisons qui peuvent être alignées avec des concepts de la source de données. Si un concept complexe contient au moins un nom qui n'a pas pu être aligné, il est systématiquement rejeté.

6.3.2 Rôle du module aligneur dans le cadre du matching complexe

Afin de répondre aux exigences particulières du matching complexe, il est judicieux de doter l'architecture du module aligneur d'INDIGO d'au moins un *aligneur de noms* et un *aligneur de contenus*. En effet, les noms extraits avec les concepts complexes peuvent facilement être erronés car leur découverte et leur intégration au sein des sources dépend de l'application de simples heuristiques. Il est donc important de prévoir un matching basé sur la comparaison de contenus de concepts pour appuyer celui faisant le rapprochement de leurs noms respectifs. L'alignement basé sur le contenu est bien sûr possible puisque le contenu de chaque combinaison peut être obtenu en combinant les valeurs (dont nous disposons) des instances de chaque concept qui la compose. La figure 6.3 illustre l'architecture du *module aligneur* d'INDIGO tel que configuré pour nos expériences du matching complexe. En plus du module superviseur, l'architecture comprend trois aligneurs de base et un coordonnateur, tous organisés hiérarchiquement. Un aperçu de chaque module est donné ci-après.

Aligneur de noms. L'*aligneur de noms* fait le rapprochement entre deux concepts complexes en comparant les noms qui les composent. La similarité entre les noms est évaluée en appliquant la métrique de similarité lexicale *JaroWinkler* [EBB+04].

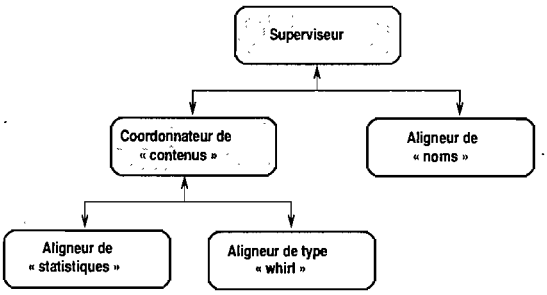


FIG. 6.3 – Le module *aligneur* d'INDIGO

Aligneur de type "Whirl". Étant donné un concept source s, l'*aligneur de type "whirl"* identifie le concept cible c dont les valeurs d'instances correspondent le plus à celles de s. Il utilise une version adaptée de la technique dite *WHIRL* qui est un type d'algorithme de classification basé sur le principe du voisin le plus proche développé par Cohen et Hirsh [CH98].

Aligneur de statistiques. L'*aligneur de statistiques*, compare, lui-aussi, le contenu de concepts. Le contenu d'un concept est représenté par un vecteur normalisé encodant sept caractéristiques, soient trois bits status définissant le type de données (*string*, *numeric* et *date*) et quatre informations statistiques (valeurs *minimum*, *maximum*, *moyenne* et *variance*). La similarité entre deux concepts est évaluée en calculant la distance euclidienne séparant les vecteur de caractéristiques décrivant leur contenu respectif.

Coordonnateur de contenus. Le *coordonnateur de contenus* combine les résultats des matchings proposés par les aligneurs qui lui sont subordonnés (i.e. aligneur de type "whirl" et aligneurs de statistiques).

6.4 Conclusion

Ce chapitre présente une solution innovatrice implémentée par INDIGO pour le matching complexe entre sources de bases de données. Évitant la recherche dans l'espace infini des combinaisons possibles de concepts, INDIGO découvre les concepts complexes en analysant les artefacts associés aux sources de données. Les concepts complexes nou-

vellement découverts sont ajoutés aux sources de données en tant que nouveaux candidats pour le matching complexe. Grâce à l'extensibilité d'INDIGO il est possible d'ajouter de nouveaux modules analyseurs et aligneurs permettant par exemple, de prendre en compte les informations intrinsèques et contextuelles des bases de données objets. Pour faciliter l'identification et le matching de concepts complexes, ces modules spécialisés pourront exploiter la logique présente dans les bases objets, la notion d'héritage, les requêtes *SQL*, etc. Qui plus est, leur intégration au sein de l'architecture d'INDIGO pourra se faire par simple configuration, sans modification additionnelle.

CHAPITRE 7

ADAPTABILITÉ D'UNE APPROCHE MULTI-STRATÉGIQUE

Pratiquement tous les systèmes actuels de l'alignement sémantique des données reposent sur une approche rigide et visent une catégorie spécifique d'applications. En effet, ils appliquent une seule et même technique de matching pour aligner tous les types de données sans tenir compte des caractéristiques particulières de chacun. Le problème est que de tels systèmes ne sont pas adaptés à la diversité et à la nature dynamique d'Internet où les sources de données sont nombreuses, très hétérogènes et en continuel changement. Ce chapitre présente l'approche adaptative de mapping implémentée par INDIGO pour réconcilier efficacement des sources de données de diverses natures. Elle se base sur une solution d'alignement multi-stratégique capable de combiner dynamiquement et intelligemment les stratégies de matching en prenant en considération les spécificités des données à aligner.

Le reste de ce chapitre se compose de deux sections principales. La première section présente l'objectif du mapping adaptatif. Quant à la seconde section, elle décrit l'architecture implémentée par INDIGO pour supporter ce type de mapping.

7.1 Objectif du mapping adaptatif

Tel que déjà discuté dans le chapitre 4 , les approches multi-stratégiques ont gagné en popularité ces dernières années et sont adoptées par un nombre croissant de systèmes de mapping [JPE01, DR02, DDH03]. Rappelons que généralement ces systèmes sont basés sur une architecture composée de plusieurs modules d'alignement indépendants qui sont supervisés par une unité centrale. Chaque module est spécialisé dans le traitement d'un type spécifique d'informations (noms de concepts, types de données, etc.) issues des schémas de données ou de leurs instances. Chaque module implémente sa propre sa stratégie d'alignement qu'il applique pour évaluer la similarité entre concepts et propose son mapping. Par exemple, un module spécialisé dans l'alignement de noms peut se baser

sur la métrique de similarité lexicale de JARO pour effectuer le rapprochement entre les concepts. Enfin, l'unité centrale du système est responsable de la combinaison des résultats fournis par les modules individuels de matching et de l'élaboration du mapping final.

Les systèmes multi-stratégiques présentent l'avantage d'être extensibles puisque l'ajout d'un nouveau module d'alignement ne requiert qu'une simple adaptation de l'unité centrale. Toutefois, la majorité de ces systèmes souffrent d'un manque d'adaptabilité car ils sont généralement conçus pour un domaine spécifique d'application. En particulier, les résultats retournés par les modules individuels d'alignement sont généralement combinés selon une procédure fixe qui est optimisée pour une application prédéfinie (e.g. pour un alignement vers un seul schéma fixe de données). La prise en charge d'un nouveau domaine nécessiterait le développement et l'application d'une nouvelle procédure de matching, puisque le système est incapable de s'y adapter automatiquement. Ainsi, la performance de tels systèmes dépend directement de la conformité des sources de données à aligner avec le domaine considéré. En somme, ils n'offrent pas plus de flexibilité que les systèmes hybrides qui combinent des stratégies individuelles de matching selon un certain algorithme prédéfini.

Pour pallier ce manque de flexibilité, nous proposons une nouvelle approche d'alignement multi-stratégique qui offre des capacités intéressantes d'adaptabilité pour un matching plus efficace. Rappelons que l'architecture du module aligneur de INDIGO se compose de plusieurs modules (coordonnateurs et aligneurs) organisés hiérarchiquement. Grâce à une technique appelée «stacking» [Wol92, TW99], chaque coordonnateur est capable de sélectionner dynamiquement et exécuter une des stratégies de matching proposées (implémentées individuellement par ses modules d'alignement subordonnés) en fonction des caractéristiques des concepts en cours d'alignement. L'originalité du module aligneur d'INDIGO réside donc dans sa capacité d'adapter sa stratégie de matching globale en fonction des caractéristques des sources de données à aligner, et ce, sans connaître les caractéristiques de ces sources au préalable. En plus, sa performance croît en fonction du nombre de stratégies de matching implémentées. En effet, le système prend automatiquement soin de n'exécuter un aligneur nouvellement ajouté que s'il s'avère être plus

efficace, pour une tâche donnée, que ses voisins dans la hiérarchie. Ainsi, étant donné les caractéristiques d'une paire de concepts à aligner, chaque coordonnateur détermine lequel parmi ses aligneurs subordonnés est le meilleur pour exécuter ce matching. Cette propriété de sélection dynamique d'une stratégie confère une plus grande flexibilité et extensibilité aux approches multi-stratégiques. Elle constitue ainsi une amélioration importante apportée aux systèmes de matching actuels.

7.2 Implémentation du mapping adaptatif dans Indigo

Étant donné un concept cible, le mapping adaptatif consiste à sélectionner, sur la base d'heuristiques, la stratégie de matching la plus pertinente qui permet de retrouver le concept source lui correspondant. Ainsi, dans l'architecture d'un module aligneur de INDIGO, c'est aux coordonnateurs qu'incombe la responsabilité du mapping adaptatif. Ils doivent être capables d'adapter la stratégie utilisée pour combiner les résultats de mapping fournis par ses aligneurs fils[1] selon les caractéristiques spécifiques des concepts à réconcilier.

En particulier, il est toujours possible d'utiliser différentes stratégies de matching pour apparier des concepts sur la base des mêmes informations sémantiques. Par exemple, plusieurs métriques de similarité lexicale [EBB+04] peuvent être utilisées pour évaluer la similarité entre des noms de concepts. La similarité de *JARO* et la distance de *Hamming* sont des exemples de telles métriques. En pratique, une stratégie peut s'avérer plus efficace que d'autres dépendamment des caractéristiques des concepts à aligner. Ainsi, les coordonnateurs doivent se baser sur des heuristiques pour décider laquelle des stratégies proposées (implémentées respectivement par leurs aligneurs fils) est la meilleure à appliquer dans chaque cas.

La section suivante explique comment la technique de stacking peut être utilisée pour fournir aux coordonnateurs les informations qui leur permettent de classer leurs aligneurs fils par rapport à l'alignement d'une paire de concepts donnée. Nous montrons

[1]Pour des raisons de simplification, les coordonnateurs et aligneurs subordonnés seront appelés aligneurs fils.

TAB. 7.1 – Valeurs de similarités proposées par trois aligneurs pour des paires de concepts source et cible

Concept source S_1	Concept cible S_2	$SimA_1$	$SimA_2$	$SimA_3$	$L(S_1, S_2)$
Name	*First_name*	0.63	0.67	0.94	1
address	*First_name*	0.45	0.32	0.27	0
...	...	...	...	...	...
Name	*Last_name*	0.52	0.98	0.84	1
Address	*Last_Name*	0.33	0.59	0.46	0
...	...	...	...	...	...

ensuite comment nous pouvons étendre la technique de stacking pour réaliser un mapping multi-stratégique adaptatif.

7.2.1 La technique de stacking

Pour être capable de sélectionner dynamiquement le meilleur aligneur parmi ses fils, un coordonnateur utilise la technique de stacking [TW99, Wol92] développée initialement dans le domaine de l'intelligence artificielle. Cette technique commence par tester (sur la base d'un certain ensemble de données d'entraînement) l'efficacité des aligneurs pour évaluer la similarité d'un concept cible donné avec d'autres concepts sources. Pour chaque concept cible, un des aligneurs est alors désigné comme étant le plus efficace pour lui retrouver un concept source correspondant. La technique de stacking a notamment permis de prouver que les classifications obtenues au niveau de sous-modèles peuvent améliorer considérablement la précision de la classification au niveau du modèle global.

La table 7.1 montre les calculs faits par l'algorithme de stacking sur un exemple simple. Pour chaque paire de concepts source et cible, chaque aligneurs (A_1, A_2 et A_3) fournit son évaluation de leur similarité. Une fonction oracle $L(S_1, S_2)$ définit si les deux concepts doivent être alignés ou non. Cette fonction retourne la valeur 1 dans le cas où S_1 correspond effectivement à S_2, et 0 dans le cas contraire. Il est à remarquer que l'alignement des données peut être assimilé à un problème de classification : chaque concept source doit être assigné à une classe dont le libellé correspond au nom d'un concept cible. Soit $s(c_i|x_j, A_k)$ une évaluation de similarité représentant la probabilité qu'un concept cible c_i corresponde à concept source x_j selon l'aligneur A_k. Et, soit $W_{A_k}^{c_i}$ le

poids assigné à l'aligneur A_k à l'égard d'un concept cible c_i. Dans notre exemple, les poids sont estimés en effectuant une régression linéaire par la méthode des moindres carrés, sur les valeurs de la table 7.1. Cette régression permet de calculer les poids minimisant l'erreur définie par la formule suivante :

$$\sum_j (l(c_i, x_j) - \sum_k s(c_i|x_j, A_k) * W_{A_k}^{c_i})^2$$

Le processus de régression maximise les poids assignés aux aligneurs proposant des valeurs élevées de similarité pour les paires de matching valides et des valeurs peu élevées ou nulles pour celles qui sont non valides. Il minimise ces poids dans le cas contraire. Par exemple, si le processus de régression aboutit à l'évaluation de poids suivante $W_{A_1}^{First\_name}$ = 0.17, $W_{A_2}^{First\_name}$ = 0.23 and $W_{A_3}^{First\_name}$ = 0.60, cela signifiera que le coordonnateur devra faire plus confiance en l'aligneur A_3 que A_2 et A_1 pour déterminer le concept source correspondant au concept cible *First_name*. Un coordonnateur a aussi bien entendu la possibilité de combiner les résultats de matching proposés par ses aligneurs fils en utilisant les poids qui leur correspondent respectivement. Ceci constitue aussi une forme de mapping adaptatif puisque la pondération varie d'un concept cible à un autre.

7.2.2 L'approche de mapping adaptatif de Indigo

Il est à rappeler que la technique de stacking est aussi utilisée par le méta-apprenant du système **LSD** (c.f. chapitre 3) pour combiner les résultats de matching renvoyés par les apprenants de base. Toutefois, tel qu'implémentée par **LSD**, l'approche de mapping dynamique suppose la présence d'un seul schéma cible fixe. Les noms de concepts composant ce schéma sont alors considérés comme les libellés de classification des concepts sources. Le fait de changer le schéma cible implique l'exécution coûteuse d'un nouveau processus de pondération des apprenants de base tenant compte des libellés des concepts du nouveau schéma. Pour surmonter ce problème, nous proposons une nouvelle approche permettant aux poids de changer dynamiquement en fonction des sources de données cibles considérées. Ceci est réalisé en utilisant les signatures de concepts comme libellés de classes au lieu de leurs noms. Une signature de concept est un vecteur de descripteurs

de propriétés prenant chacun la valeur vrai ou faux. Ainsi, une signature de concept permet de caractériser un ensemble de concepts partageant les mêmes valeurs de propriétés. Dans le cadre du stacking, l'emploi de signatures s'avère tout à fait pertinent sachant que les décisions de matching se basent justement sur la comparaison des caractéristiques de concepts. Il est toutefois important que les signatures soient composées de descripteurs significatifs pour le coordonnateur et qu'elles puissent influencer favorablement la performance des stratégies de matching qui lui sont associées. Afin d'expliquer la façon dont sont choisis les descripteurs de la signature d'un concept, considérons un coordonnateur supervisant deux aligneurs fils. Le premier aligneur applique la similarité de **JARO** pour comparer les noms de concepts, tandis que le second implémente une stratégie d'apprentissage similaire à celle de l'apprenant de noms dans le système **LSD** (cf. chapitre 3.

- La similarité de **JARO** évalue la similarité de deux chaînes de caractères en tenant compte du nombre et de l'ordre des caractères qu'elles partagent.
- L'apprenant de noms de **LSD** est un module chargé de la réconciliation des noms de concepts. Comme l'alignement est fait entre des schémas **XML**, chaque concept est identifié par son chemin complet de noms. Pour constituer un ensemble de données d'entraînement, tous les noms de balises composant un chemin XML (i.e. représentant le nom d'un concept) sont tout d'abord étendus par leurs synonymes. Chaque chemin étendu est ensuite manuellement associé, par un expert utilisateur, à un libellé de concept cible. Suit alors la phase d'alignement. Les nouveaux chemins de noms sont alors classifiés en utilisant la technique **WHIRL** développée par Cohen et Hirsh [CH98], qui compare ces chemins avec ceux étendus de la base d'entraînement. La mesure **TF/IDF** est utilisée pour évaluer la similarité entre les chemins de noms. Les paires de chemins de noms sont triés par ordre décroissant de similarité et seulement ceux dépassant à certain seuil prédéfini sont retenus. Un nouveau chemin de noms est enfin associé au libellé du concept cible (dans l'ensemble des données d'entraînement) auquel il est le plus similaire.

Comme on peut deviner, l'approche de **LSD** ne peut pas être assez efficace pour aligner des noms possédant plusieurs synonymes. Souvent peu pertinents dans le contexte

du domaine des sources de données à aligner, ces synonymes sont susceptibles de générer du bruit lors du processus de mapping. Inversement, basée sur la comparaison des formes lexicales des noms, la similarité de **JARO** peut s'avérer une solution robuste dans des situations où **LSD** est faible. En revanche, il est plus probable que l'approche **LSD** soit plus efficace pour aligner des noms lexicalement différents bien que partageant une même sémantique, et ce, surtout si le volume des données d'entraînement devient important.

Dans l'exemple dont nous venons de discuter, deux caractéristiques de concepts peuvent être identifiés comme facteurs influençant l'efficacité des deux stratégies de matching proposées, soient la métrique de JARO et la technique d'apprentissage de noms. Ces caractéristiques sont 1) le nombre de synonymes et 2) le status d'abréviation d'un concept. En effet, ces deux propriétés peuvent être utilisées comme descripteurs composant la signature des concepts si l'on considère les deux types d'aligneurs de notre exemple.

- **Degré de synonymie** : Le nombre de synonymes (NS) qui sont associés à un nom de concept peut être déterminé en consultant un dictionnaire externe tel que **WordNet**. Un seuil limite s peut être utilisé pour évaluer le degré de synonymie d'un concept. Si $NS < s$, alors le nombre de synonymes est considéré peu élevé et le degré de synonymie est alors évalué à 0. Sinon, si $NS \geq s$, le degré de synonymie est considéré élevé et est par conséquent évalué à 1.
- **Status d'abréviation** : Cette valeur indique si le nom de concept est abrégé (= 1) ou non (= 0).

Ainsi, les noms de concepts peuvent dans ce cas être décrits par une signature composée de deux descripteurs (degré de synonymie, status d'abréviation). Puisque chaque descripteur est évalué à 0 ou 1, quatre signatures possibles de classes de concepts peuvent être définies : $(0,0)$, $(0,1)$, $(1,0)$ et $(1,1)$. Si en plus, on suppose que les noms abrégés ne peuvent pas avoir de synonymes, alors seules les classes $(0,0)$, $(0,1)$ et $(1,0)$ peuvent être prises en compte. L'ensemble de toutes les signatures possibles de classes est appelé l'ensemble global de classes de concepts et est noté CG. En général, si n descripteurs sont définis et aucune contrainte n'est imposée sur leurs valeurs (0 ou 1) respectives, alors la cardinalité de CG est 2^n. Les deux définitions suivantes seront utilisées pour expliquer

le *processus de stacking* basé sur les signatures de classes.

Configuration de classes (CC) : Une CC est un sous-ensemble de CG. Elle est obtenue en fixant la valeur d'un des descripteurs. Par exemple, si la source de données cible ne contient aucun nom de concept abrégé, alors les coordonnateurs ont besoin de considérer une CC composée uniquement des signatures des deux classes cibles $(0, \underline{0})$ et $(1, \underline{0})$ (le second descripteur est fixé à 0). Cette transformation peut être vue comme une extraction d'un espace de dimension $n-1$ dans un espace vectoriel de dimension n.

Compatibilité entre une configuration de classes et une source de données : Une *configuration de classes* CC est dite compatible avec une source de données S si chaque classe dans CC correspond au moins à un concept de S et si pour chaque concept de S il existe une classe dans CC à laquelle il est associé. Par exemple, la configuration $\{(0,0),(0,1)\}$ n'est pas compatible avec une source de données qui ne contient pas de nom abrégé puisque la classe $(0,1)$ ne peut être associée à aucun concept de cette source de données. D'un autre côté, elle n'est pas compatible avec une source de données contenant un ou plusieurs concepts dépassant le degré limite de synonymie, puisque ces concepts ne retrouveront pas de classe correspondante (i.e. $(1,0)$) dans l'ensemble $\{(0,0),(0,1)\}$.

Une configuration de classes peut être récursivement construite à partir d'un CG. Soit $setCC_i(CG)$ l'ensemble des configurations de classes du niveau i (i.e. l'ensemble des CC obtenus en fixant i descripteurs dans les signatures de classes de CG). Ainsi, au niveau 0 nous avons $setCC_0(CG) = \{CG\}$. Au niveau i, chaque $CC \in setCC_i(CG)$ contient $2^{(n-i)}$ classes. La cardinalité de $setCC_i(CG)$ est $\binom{n}{i} * 2^i$ soit le nombre de façons de choisir i descripteurs fixes parmi n, multiplié par le nombre de façon de fixer les valeurs de ces i descripteurs. Par conséquent, étant donné n descripteurs, le nombre total de CC possibles est $\sum_{i=0}^{n} 2^i \frac{n!}{i!(n-i)!}$. Étant donné deux descripteurs, 9 CC sont donc possibles.

Il est facile de démontrer qu'une source de données ne peut pas être compatible avec deux configurations de classes distinctes CC_i et CC_j (avec $i \neq j$). Comme nous le savons, chaque CC définit des valeurs fixes pour une combinaison différente de descripteurs. Soit d un descripteur fixé à 0 (resp. 1) par CC_i et non fixé par CC_j. Ceci veut dire que CC_j contient une signature de classe pour laquelle $d = 1$ (resp. $d = 0$). Si une source de données est compatible avec CC_j, c'est qu'elle contient un concept pour lequel $d = 1$

$CC_{new} \leftarrow getCC(CG, S)$
$found \leftarrow setKnownCC.search(CC_{new})$
$if\ (found)\ then\ \{$
    $setAlignersWeigths \leftarrow setKnownCC.getWeights(CC_{new})$
    $finalMapping \leftarrow ApplyWeightedCombination(setAlignersWeigths, setMappings)$
$\}$
$else\ \{$
    $finalMapping \leftarrow ApplyAverageCombination(setAlignersWeigths, setMappings)$
    $setKnownCC.add(CC_{new})$
$\}$
$setKnownCC.reassessWeights(CCnew, setMappings, referenceMapping)$

FIG. 7.1 – Pseudo-code montrant la procèdure de stacking appliquée par un coordonnateur pour un alignement vers une source de données cible S

(resp. $d = 0$) et par conséquent ne peut être décrit par aucune des signatures de classes dans CC_i.

En pratique, chaque coordonnateur doit maintenir une ou plusieurs CC qu'il enregistre au fur et à mesure qu'il rencontre différents types de sources de données. Ci-après, nous expliquons comment les CC sont utilisés lors d'un processus d'alignement (cf. figure 7.1 pour un pseudo-code relatif à ce processus) :

- Le coordonnateur analyse la source de données cible et détermine le sous-ensemble de signatures de CG avec lequel elle est compatible. Pour cela, il détermine la signature de chaque concept de la source de données en fonction des descripteurs prédéfinis. L'ensemble des signatures ainsi obtenues sont stockées dans CC_{new}
- CC_{new} est recherché dans l'ensemble $setKnownCC$ contenant toutes les CCs que le coordonnateur a déjà enregistrées. Si $CC_{new} \in setKnownCC$, alors CC_{new} est une configuration connue. On s'y réfère donc pour sélectionner la stratégie (la meilleure à ce jour) à exécuter pour aligner un concept cible. Cette stratégie utilise les poids estimés lors du dernier processus de stacking pour la configuration CC_{new}. Si $CC_{new} \notin setKnownCC$, une stratégie de combinaison (e.g. la moyenne) est appliquée par défaut pour tous les concepts à aligner.
- À la fin du processus d'alignement, les poids sont réévalués pour CC_{new} dans le cas où il s'agissait d'une configuration déjà connue. Sinon, CC_{new} est ajouté à $setKnownCC$, et de nouveaux poids sont calculés en fonction des performances

respectives des stratégies d'alignement.

La technique d'apprentissage décrite ci-haut a l'avantage majeure de conférer à chaque coordonnateur la capacité d'améliorer sa propre performance au fur et à mesure qu'il rencontre de nouvelles sources de données à aligner. En effet, les poids associés aux classes de concepts dans une CC donnée sont réévalués après chaque exécution de mapping avec l'effet d'améliorer continuellement la qualité de l'alignement global. Sur le plan de la prise en charge de la diversité des sources, on constate que l'apprentissage réalisé pour une CC donnée peut ainsi profiter à toute autre source de données qui lui est compatible.

7.3 Conclusion

Ce chapitre présente une approche innovante pour implémenter le mapping dynamique. Se basant sur une adaptation de la technique du stacking chaque coordonnateur est capable de sélectionner et de pondérer dynamiquement les résultats des aligneurs fils les plus appropriés pour déterminer le correspondant d'un concept cible donné en fonction de ses caractéristiques. Qui plus est, en décrivant les concepts par une signature de leurs caractéristiques, les coordonnateurs peuvent «apprendre» l'alignement de certaine classe de configuration de signatures de concepts cibles et ainsi améliorer leur performance au fil du temps. Finalement, on notera que le processus d'apprentissage a du même coup l'avantage de ne pas se limiter à une source de données cible prédéterminée.

CHAPITRE 8

TESTS ET RÉSULTATS EXPÉRIMENTAUX

Les tests expérimentaux ont concerné nos trois contributions majeures dans le domaine de l'alignement sémantique des données : (1) l'alignement lexical basé sur le contexte (chapitres 5), (2) le mapping complexe (chapitre 6) et (3) le matching dynamique (chapitre 7). Ils se sont divisés essentiellement en deux catégories principales. La première catégorie de tests a portée sur l'évaluation de l'impact de la contextualisation des sources de données à aligner et de l'alignement dynamique sur la performance globale d'INDIGO. Quant à la seconde catégorie, elle visait plutôt la comparaison de l'efficacité d'INDIGO avec celle de systèmes souvent cités dans la littérature et dont les objectifs sont similaires aux nôtres.

Le reste du présent chapitre s'articule autour de quatre sections. La première section présente l'outil INDIGO que nous avons développé et utilisé pour valider expérimentalement notre approche. Les trois autres sections décrivent les résultats des tests relatifs, respectivement, à nos trois principales contributions de recherche.

8.1 Indigo : l'outil

L'outil INDIGO consiste en une plateforme d'alignement sémantique de données qui se compose de deux modules principaux :

1. Le module analyseur de contexte : le module analyseur de contexte se compose d'un ensemble de programmes chargés de l'enrichissement des sources de données à aligner. Ces programmes sont exécutés en mode différé et interviennent dans une phase précédant le mapping proprement dit de ces sources de données.
2. Le module d'alignement : le module d'alignement d'INDIGO permet entre autre la configuration du module aligneur et la prise en charge du processus d'alignement des données.

Le langage de programmation JAVA a été exclusivement utilisé pour développer l'ensemble des modules composant INDIGO. Par ailleurs, notons que, pour le moment, seules les sources de données décrites sous un format XML (e.g. XSD, DTD, RDF, OWL etc.) sont supportées par INDIGO. Les autres types de modèles (e.g. modèle relationnel) doivent actuellement être convertis vers ce format avant de pouvoir être analysés par INDIGO.

8.1.1 Le module analyseur de contexte

Le module analyseur de contexte s'exécute sur une ligne de commande. Il reçoit comme paramètres en entrée le nom de la source de données à enrichir et le chemin menant vers le répertoire qui contient tous les fichiers composant le contexte de cette dernière. Il génère en sortie deux fichiers de concepts voisins. Le premier fichier CC-Doc (cf. chapitre 5), associe chaque concept de la source de données en entrée avec un ensemble de noms trouvés près de lui dans l'un ou l'autre des documents composant son contexte descriptif. Le second est un fichier de concepts complexes listant des combinaisons de concepts extraites du contexte opérationnel (cf. chapitre 6). Pour ce faire, le module analyseur de contexte s'appuie sur ses deux principales composantes : le collecteur de noms de concepts et l'extracteur des concepts complexes. Ces deux modules peuvent eux aussi être exécutés directement à partir d'une ligne de commande. Pour cela, ils doivent recevoir comme paramètre d'entrée la liste des fichiers composant le contexte descriptif pour le cas du collecteur, et celle des artefacts formant le contexte opérationnel pour le cas de l'extracteur des concepts complexes. Les fichiers de concepts voisins générés ont tous les deux *.ctx* comme extension.

Le nom du fichier CC-Doc se compose du nom de la source de données correspondante suivi du suffixe *_lex* (i.e. *NomSourceDeDonnees_lex.ctx*). Le nom du fichier des concepts complexes diffère de celui de CC-Doc par son suffixe *_cpx*. Une fois générés, ces fichiers sont mis dans un répertoire prédéfini pour que le module aligneur puisse facilement les retrouver et les utiliser lors du processus de mapping.

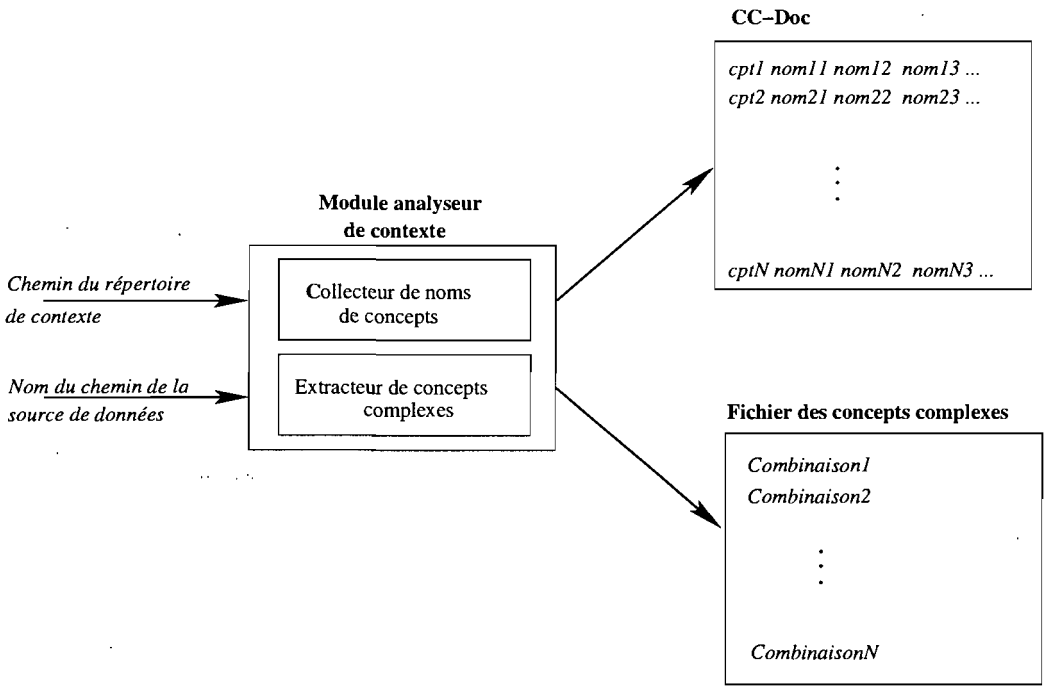


FIG. 8.1 – Exécution du module analyseur de contexte

8.1.2 Le module intéractif d'alignement

Le module d'alignement d'INDIGO offre une plateforme configurable assistant l'utilisateur dans l'exécution du processus de mapping de sources de données après leur contextualisation par le module analyseur de contexte. Toutefois il est à noter que ce module est aussi capable d'aligner des sources de données non contextualisées ou de tenir compte du contexte de seulement l'une d'entre elles. Par ailleurs, il offre une interface composée de deux écrans principaux. Le premier écran permet entre autre la configuration du module aligneur, le chargement des sources de données à réconcilier et l'exécution du mapping. Quant au second écran, il permet la saisie manuelle d'un mapping de référence et l'évaluation de la qualité de l'alignement généré par l'aligneur d'INDIGO tel que configuré sur le premier écran. Les deux paragraphes suivants décrivent plus en détail le rôle de chacun de ces deux écrans.

8.1.2.1 Écran d'alignement

L'écran d'alignement (cf. figure 8.2) est subdivisé en deux sections : une section entête et une section principale.

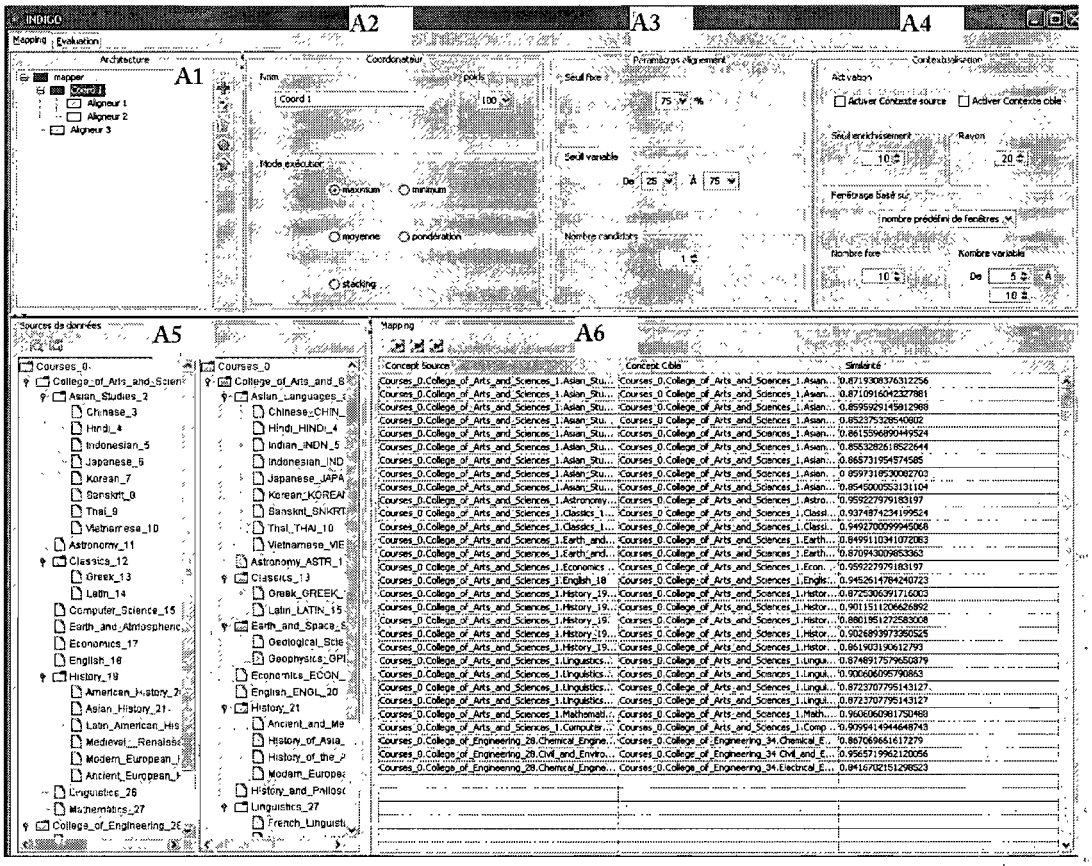


FIG. 8.2 – Écran d'alignement de INDIGO

La section entête est réservée pour la configuration du module aligneur et la définition des paramètres d'alignement. Elle se compose de quatre zones de configuration séparées comme suit :

1. Zone de configuration de l'architecture du module aligneur (zone A1 sur la figure) : la zone de configuration de l'architecture du module aligneur offre une barre d'outils permettant de construire dynamiquement la structure arborescente de cette architecture. Pour cela, elle permet de pointer sur un noeud dans la hiérarchie et

de lui créer un ou plusieurs fils ou bien de le supprimer. La barre d'outil offre aussi la possibilité d'ouvrir un écran pour le chargement de l'une des architectures déjà créées.

2. Zone de configuration des coordonnateurs et aligneurs (zone A2 sur la figure) : la zone de configuration des coordonnateurs et aligneurs affiche soit un écran de configuration de coordonnateur ou bien celui d'un aligneur selon que l'utilisateur sélectionne un noeud interne ou une feuille au niveau de l'architecture créée dans la zone A1. L'écran de configuration d'un coordonnateur permet de définir son nom, de lui associer manuellement un poids (allant de 1 à 100) et de définir le mode de combinaison des résultats de matching de ses aligneurs subordonnés. Les modes de combinaison supportés actuellement sont le maximum, le minimum, la moyenne, la pondération et le stacking. La pondération consiste à utiliser les poids attribués manuellement par l'utilisateur à chaque composante de l'architecture. Quant à l'écran de configuration d'un aligneur, il permet de saisir son nom et sa description, de lui attribuer un poids et de lui associer une stratégie de matching dont le choix se fait à partir d'une librairie de statégies prédéfinies. Il est à noter que les classes implémentant les stratégies disponibles dans la librairie descendent toutes d'une même classe mère appelée *Strategy*. Ainsi, elles héritent, entre autres, des méthodes implémentées par cette dernière pour l'accès et la navigation à travers les sources de données à aligner et leur contexte.

3. Zone de définition des paramètres d'alignement (zone A3 sur la figure) : la zone de définition des paramètres d'alignement permet de définir la valeur d'un seuil fixe qui sera utilisé lors de l'alignement pour ne retenir que les paires de concepts dont la valeur de similarité lui est supérieure. Par ailleurs, cette zone permet aussi de définir un seuil variable borné par des valeurs minimum et maximum. L'exécution d'un alignement basée sur un seuil variable consiste à rechercher le seuil entre ces deux valeurs qui permet de générer le meilleur mapping. Enfin, cette zone A3 offre aussi la possibilité de définir le nombre maximum de candidats à proposer pour chaque concept cible. Lors de nos expériences ce paramètre a toujours été fixé à 1.

4. Zone de configuration de contexte (zone A4 sur la figure) : la zone de configuration de contexte permet la définition des paramètres liés à la contextualisation des sources de données. En effet, elle permet à l'utilisateur d'indiquer si le contexte de l'une ou des deux sources de données à aligner doit être pris en compte lors de l'alignement. La zone A4 permet également de fixer le seuil d'enrichissement sémantique limitant le nombre de concepts d'enrichissement à associer à chaque concept d'une source de données. Par ailleurs, elle offre également la possibilité de fixer le rayon contextuel (i.e. la distance séparant un concept de la source de données du nom le plus éloigné dans une liste d'enrichissement en allant vers la droite ou la gauche). Les noms situés en dehors de ce rayon contextuel peuvent ainsi être exclus dans le processus d'enrichissement. Enfin, cette zone donne à l'utilisateur la possibilité de choisir la façon de subdiviser le GDC d'un concept (cf. chapitre 5) : par un nombre fixe ou variable de boîtes. Dans ce dernier cas, l'utilisateur définit la limite inférieure et supérieure du nombre possible de boîtes à utiliser pour subdiviser un GDC. Dans un alignement contextuel basé sur un nombre variable de boîtes, le module aligneur choisira le nombre compris entre ces deux limites qui permet de générer le meilleur mapping.

Quant à la section principale, elle se subdivise en deux zones :

1. Zone des sources de données (zone A5 sur la figure) : la zone des sources de données permet le chargement des sources de données à aligner. Ces dernières sont alors affichées sous forme d'arborescences au niveau de deux sous-zones. Chacune peut y être consultée en naviguant à travers sa structure.
2. Zone de mapping (zone A6 sur la figure) : la zone de mapping est réservée à l'exécution de l'alignement et à la visualisation de son résultat (i.e. liste des paires de concepts retenues associées à leur similarité).

8.1.2.2 Écran d'évaluation

L'écran d'évaluation (cf. figure 8.3) d'INDIGO se compose des deux zones principales suivantes :

1. La zone *Mapping de référence* (zone E1 sur la figure) : la zone *Mapping de référence* permet la saisie manuelle d'un mapping de référence. Pour faciliter cette saisie, la liste des concepts cibles est automatiquement générée dans le côté droit de cette zone au premier chargement de la source de données cible. En plus, la sélection de chaque case correspondant à un concept source fait apparaître automatiquement un menu déroulant listant l'ensemble des concepts sources parmi lesquels l'utilisateur peut faire un choix pour l'associer à un concept cible donné.
2. La zone *Mesures de performance* (zone E2 sur la figure) : la zone *Mesures de performance* affiche la valeur des quatres métriques utilisées pour évaluer la qualité du mapping généré à partir de l'écran d'alignement, en le comparant au mapping de référence. Les mesures utilisées concerne la précision, le rappel, la f-mesure et la mesure overall [MGMR02]. Cette dernière mesure évalue l'effort nécessaire pour corriger un alignement proposé en éliminant les faux positifs et ajoutant les vrais négatifs (i.e. $overall = rappel * (2 - \frac{1}{précision})$). Elle n'a toutefois pas été directement utilisée dans nos évaluations de la performance de INDIGO.

8.2 Tests relatifs au mapping lexical intégrant le contexte

Nous avons mené deux types de tests pour évaluer le mapping lexical intégrant le contexte des sources de données à apparier. Dans la première série de tests, nous avons évalué l'impact de l'enrichissement de noms des concepts composant ces sources de données sur la performance du matching lexical. Ce matching était ici basé des techniques utilisant des métriques de comparaison de chaînes de caractères. La seconde série de tests a eu pour objectif de comparer la performance d'INDIGO avec celle des systèmes SF (Similarity Flooding), V-doc et Cupid qui sont considérés comme de bonnes références dans le domaine de l'alignement sémantique. Nous avons pu télécharger le code source de l'algorithme SF [Mel03] et du système V-doc [Gro06] à partir de leurs sites web respectifs. Pour le cas de Cupid, nous avons dû implémenter l'algorithme nous-même car aucune implémentation n'était disponible.

Rappelons enfin que pour le cas des tests concernant le mapping lexical intégrant le

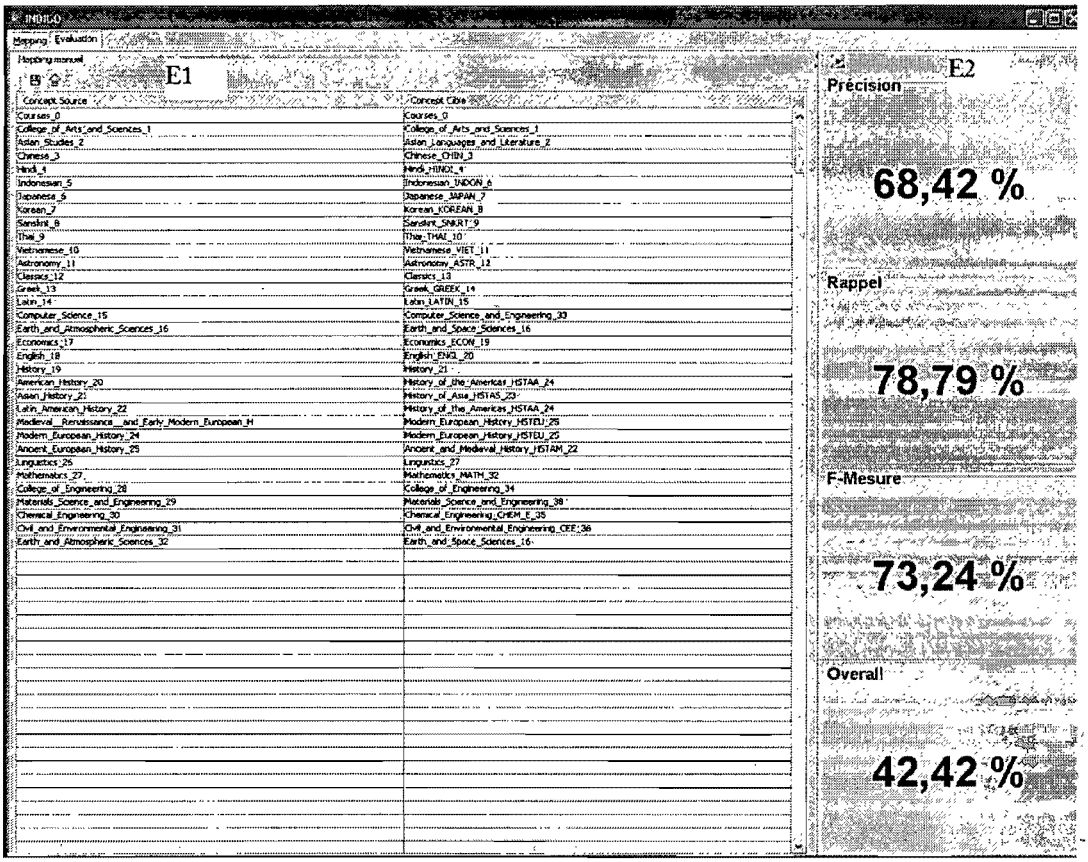


FIG. 8.3 – Écran d'évaluation de INDIGO

contexte, deux études de cas ont été utilisées pour leur validation (cf. chapitre 5) :

- Alignement de deux schémas de bases de données issus respectivement des applications *Java Pet Store* [Mic05] et *eStore* [McU03].
- Alignement de deux ontologies décrivant les programmes de cours donnés à l'Université de Cornell et à l'Université de Washington [ACC04].

8.2.1 Impact de la contextualisation

Pour tester l'impact de la contextualisation des sources de données sur la performance du mapping lexical, le module aligneur a été configuré avec un aligneur de noms supportant à chaque exécution l'une des cinq métriques de comparaison de chaînes de

	sans contexte			avec contexte			variation		
	prec.	*rap.*	*f-m.*	*prec.*	*rap.*	*f-m.*	*prec.*	*rap.*	*f-m.*
Jaro-Winkler	25.00	33.33	28.57	40.00	50.00	44.44	+60%	+50%	+56%
Smith-Waterman	23.81	41.67	30.30	66.67	66.67	66.67	+180%	+60%	+120%
Levenshtein	75.00	25.00	37.50	62.50	41.67	50.00	−17%	+67%	+34%
Needleman-Wunch	37.50	25.00	30.00	46.67	58.33	51.85	+24%	+133%	+73%
Q-Gram	100.00	25.00	40.00	60.00	50.00	54.55	−40%	+100%	+36%
Average	52.26	30.00	33.27	55.16	53.34	53.50	+42%	+82%	**+64%**

TAB. 8.1 – Variations de performances due à la contextualisation des sources de données observée dans l'alignement de *mini-PetStore* avec *mini-eStore* en utilisant INDIGO

caractères Jaro-Winkler, Smith-Waterman, Levenshtein, Needleman-Wunch et Q-Gram présentées dans la section 5.1.1. Un test préliminaire a été mené sur une version simplifiée de la première étude de cas que nous avons appelée *mini-PetStore/mini-eStore*. Cette forme réduite d'étude de cas contient seulement les parties des deux modèles de données concernant les entités *produit*, *catégorie*, *article* et *inventaire*. Ce test nous a surtout permis d'observer le comportement de nos formules de similarités et de fixer les valeurs des divers paramètres qui y sont liés (i.e. w, λ, ...). Chaque configuration du module aligneur a été séparément évaluée avec et sans prise en compte de la contextualisation des sources de données à réconcilier. La performance de chaque mapping généré a été évaluée en utilisant la *f-mesure* qui présente l'avantage de combiner les valeurs de la *précision* et du *rappel* (i.e. $f\text{-}mesure = \frac{2 \star précision \star rappel}{précision + rappel}$). Par ailleurs, le mapping de référence qui a été utilisé comme base pour évaluer les mesures de performance, a été défini manuellement.

La table 8.2.1 décrit les résultats obtenus en fixant le seuil d'enrichissement à $K = 20$ et le paramètre de subdivision de GDC à $w = 10$. La constante λ a été fixée à 4. Les résultats montrent que la contextualisation des sources de données a un impact positif sur la performance des aligneurs de noms. Elle a permis d'améliorer la *f-measure* par 64% globalement. Dans le cas particulier de la distance *Smith-Waterman*, la *f-measure* du processus d'alignement a été augmentée de 120%. Dans les autres cas, l'amélioration peut sembler ne pas être aussi spectaculaire, mais en interprétant ces résultats, il faut tenir compte 1) du fait que pour chaque test de mapping, un seul aligneur a été utilisé, et 2) de l'importante hétérogénéité entre les deux sources de données alignées. En effet,

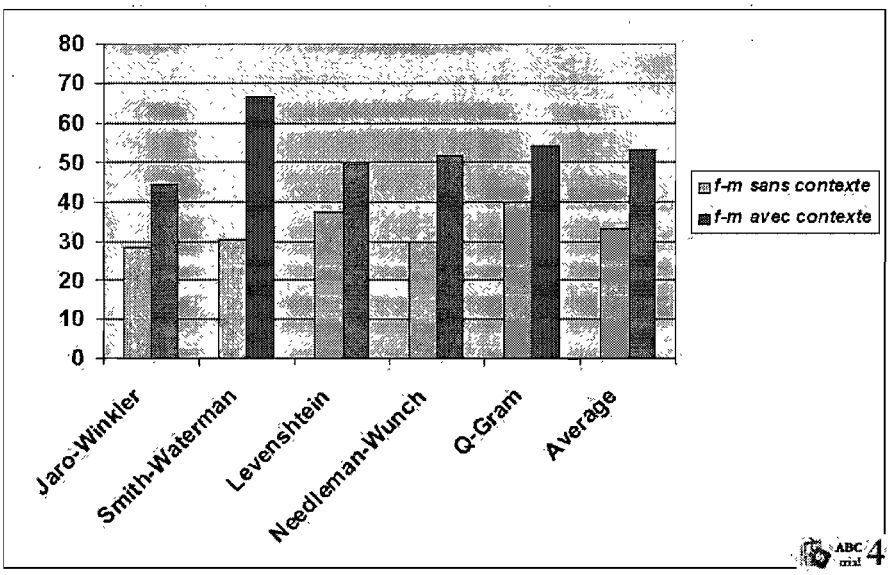


FIG. 8.4 – Graphique illustrant les résultats de variations de performances due à la contextualisation des sources de données observée dans l'alignement de *mini-PetStore* avec *mini-eStore* en utilisant INDIGO

PetStore et *eStore* présentent deux modèles de données très différents : 52 paires de matching ont été découvertes manuellement entre les sources de données *PetStore* et *eStore* sur un total de 9396 (i.e. 108*87) paires possibles. Ce rapport donne une idée du degré d'hétérogénéité entre ces deux sources de données.

Enfin, il est important de noter que contrairement à la *précision*, la mesure de *rappel* s'est améliorée dans tous les cas d'exécutions grâce à la contextualisation. Ainsi, tant que la valeur de la *précision* ne diminue pas, cela signifie que de nouveaux matchings vrais positifs seraient découverts. Enfin, remarquons que grâce à la contextualisation des sources de données les mesures de *rappel* et *précision* ont pu être améliorées respectivement de 42% et 82% sur l'ensemble des métriques utilisées.

8.2.2 Comparaison d'Indigo avec d'autres systèmes de mapping

Comme déjà mentionné, INDIGO a été mis à l'épreuve face aux systèmes SF, V-Doc et Cupid. Ce test de comparaison a lui aussi visé l'évaluation de l'impact de la

	mini-PetStore/mini-eStore			PetStore/eStore			mini-cornell/mini-washington			cornell/washington		
	prec.	*rap.*	*f-m.*	*prec.*	*rap.*	*f-m.*	*prec.*	*rap.*	*f-m.*	*prec.*	*rap.*	*f-m.*
SF	50.00	25.00	33.33	41.66	14.92	21.97	1.0	79.41	88.52	60.0	72.22	65.54
V-Doc	100.00	7.00	13.33	66.66	2.98	5.71	65.62	61.76	63.63	17.36	46.29	25.25
Cupid	40.00	16.67	23.53	19.40	30.95	23.85	85.29	87.88	86.87	48.68	69.81	57.36
INDIGO(1)	37.50	25.00	30.00	29.41	11.9	16.95	86.67	78.79	82.54	62.75	60.38	61.54
INDIGO(2)	80.00	33.33	47.06	32.65	38.10	35.16	93.33	84.85	88.89	69.64	73.58	71.56

TAB. 8.2 – Comparaison de la performance d'INDIGO avec les résultats de matching des systèmes SF, V-Doc et Cupid

contextualisation des sources de données à aligner sur l'efficacité d'INDIGO. Mais cette fois-ci, nous avons surtout surveillé l'évolution de la performance d'INDIGO par rapport à celle des autres systèmes. Le module aligneur a été configuré pour ne comporter qu'un seul aligneur implémentant une stratégie de matching basée sur la métrique *Q-Gram*. Nous avons adapté la stratégie pour qu'elle procède à la normalisation des noms de concepts avant de les aligner. Chaque système a été exécuté sur les deux études de cas pour générer ainsi : (1) le mapping entre les sources de données *PetStore* et *eStore*, et (2) le mapping entre les ontologies des cours de l'*Université de Cornell* et de l'*Université de Washington*. L'expérience a été menée sur les deux versions des schémas de données liés aux études de cas : forme réduite et originale. Par ailleurs, la syntaxe des sources de données a été adaptée aux formats requis en entrée par chaque système évalué. Le format XSD a ainsi été généré pour l'algorithme SF tandis que pour le système V-Doc les données en entrée ont été transformées en le format RDF. Quant à l'algorithme de Cupid, aucune conversion de format n'a été requise. En effet, nous avons implémenté cet algorithme de sorte à ce qu'il supporte le format original des sources de données (i.e. XML).

Pour chaque alignement, la *f-measure* a de nouveau été calculée. La table 8.2.2 regroupe les résultats obtenus suite à l'exécution des systèmes sur les deux études de cas. En particulier, les deux dernières lignes présentent les performances réalisées par INDIGO pour des alignements faits respectivement (1) sans et (2) avec intégration de contexte.

Comme nous pouvons le remarquer, grâce à la prise en compte du contexte des sources de données, la performance de INDIGO a pu être notablement améliorée, et

	SF	V-Doc	Cupid	INDIGO(1)	INDIGO(2)
précision	62.91	62.41	48.34	54.08	68.90
rappel	47.89	29.51	51.32	44.01	57.46
f-mesure	52.34	26.98	47.90	47.75	60.67

TAB. 8.3 – Comparaison de la performance globale de INDIGO avec les résultats de matching de SF, V-Doc et Cupid

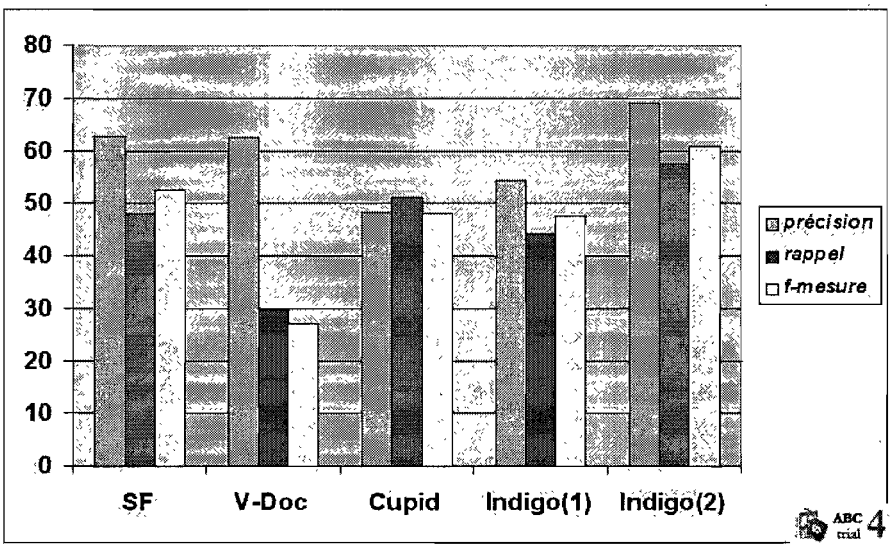


FIG. 8.5 – Graphique illustrant les résultats de comparaison de la performance globale de INDIGO avec les résultats de matching de SF, V-Doc et Cupid

ce, pour toutes les études de cas. Cette observation est importante dans la mesure où l'amélioration de la performance de INDIGO a été effective dans l'alignement de deux modèles différents de sources de données. En effet, l'étude de cas *PetStore/eStore* concerne des sources de bases de données tandis que celle de *Cornell/Washington* se réfère à des ontologies. Ceci confirme donc que la contextualisation peut être rentable dans l'alignement de différents modèles de données.

Par ailleurs, grâce à la contextualisation des sources de données alignées, INDIGO a réussi à dépasser tous les autres systèmes. De plus, sa performance globale a pu être améliorée de 27% (cf. table 8.2.2), ce qui lui a permis de passer du troisième rang au

premier dans le classement des quatre systèmes évalués. La contextualisation a toutefois requis un certain effort supplémentaire pour le pré-traitement de tous les artefacts composant le contexte des sources de données. Dans la première étude de cas, 7 documents au total composant les contextes descriptifs de *eStore* et *Java Pet Store* ont été analysés par INDIGO. Ces documents ont été découpés en 1600 phrases à peu près : 1006 phrases ont été générées à partir du contexte descriptif de *eStore* et 623 à partir de celui de *Java Pet Store*. Quant à la seconde étude de cas, 770 descriptions de cours ont été extraites à partir des catalogues en ligne des cours des Universités de Cornell et de Washington. À peu près 8000 phrases ont été analysées à partir du site web de l'Université de Cornell et 9400 de celui de l'Université de Washington. La charge supplémentaire liée à l'analyse d'un contexte dépend de la taille de celui-ci ainsi que du nombre de concepts composant la source de données associée. Nous pensons que malgré cette charge additionnelle, la contextualisation reste un investissement rentable surtout dans le cas de sources de données très hétérogènes. L'amélioration spectaculaire de la performance de INDIGO grâce à la contextualisation dans le cas de l'alignement de la paire de sources de données *PetStore/eStore* (i.e. +100%) confirme ce fait.

Enfin, remarquons que V-Doc a réalisé une assez faible performance comparativement aux autres systèmes. Ceci est vraisemblablement dû à l'absence des annotations dans les sources de données utilisées dans nos tests. En effet, le processus d'alignement de V-Doc se base fondamentalement sur ces annotations. Il crée pour chaque concept un document virtuel composé de mots extraits de son nom ainsi que des commentaires et annotations qui lui sont associées au sein d'une ontologie (cf. section 3.4).

8.3 Expérimentation du mapping complexe

Comme déjà mentionné dans le chapitre 6, Les expériences relatives au mapping complexe ont porté sur les quatre applications de vente d'animaux en ligne suivantes : *Java Pet Store* [Mic05], *eStore* [McU03], *PetMarket* [Ado07] et *PetShopDNG* [Dot03]. À l'instar de *Java Pet Store* et *eStore*, les deux autres applications ont pu aussi être téléchargées à partir d'Internet. Les tests que nous avons conduits se sont scindés en

TAB. 8.4 – Résultats de la performance de l'*analyseur de contexte*.

Appli-cation	tables	concepts	concepts complexes	concepts complexes extraits			Performance	
				total	cohésifs	pertinents	cohésion	pertinence
PetStore	13	86	7	40	32	5	80%	71%
eStore	15	73	3	23	18	3	78%	100%
PetShop	5	41	2	8	7	2	87%	100%
PetMarket	11	71	3	11	8	2	73%	67%
total	44	271	15	82	65	12	79%	**80%**

deux parties. Une première série de tests s'est intéressée à l'évaluation de la performance de l'analyseur de contexte en terme d'extraction de concepts complexes pertinents. La seconde série de tests a quant à elle mis le focus sur la performance du mapping complexe proprement dit.

8.3.1 Performance de l'exploration du contexte

Deux mesures, nommées respectivement *cohésion* et *pertinence*, ont été définies pour évaluer la performance de l'*analyseur de contexte* de INDIGO.

- *Cohésion.* Un concept complexe est dit cohésif s'il présente une combinaison de concepts qui soit sémantiquement correcte. Par exemple, un concept complexe qui est associé à la combinaison de concepts *concat(first_name, unit_price)* ne peut pas être considéré cohésif. En effet, la concaténation du prénom d'une personne avec le prix unitaire d'un produit n'a pas de signification communément acceptée. La mesure de cohésion indique le pourcentage des concepts complexes extraits qui sont effectivement cohésifs.
- *Pertinence.* Un concept complexe est dit pertinent s'il apparaît dans le mapping de référence. Étant donné deux sources de données à aligner, la mesure de pertinence calcule donc la proportion des concepts complexes présents dans le mapping de référence, parmi tous ceux qui ont effectivement été extraits par l'*analyseur de contexte*.

Pour éliminer le bruit et améliorer la performance, l'*analyseur de contexte* recourt à deux filtres. Le premier filtre (introduit dans le paragraphe 6.3.1.2.2) vise à ne garder que les combinaisons pour lesquelles chacun des concepts a pu trouver son correspondant

dans la source de données (i.e. avec une similarité supérieure au seuil prédéfini). Quant au second filtre, celui-ci rejette simplement toutes les combinaisons dont le nombre de concepts excède un certain seuil. Ceci est nécessaire car les combinaisons très larges sont souvent peu cohésives. Il faut toutefois noter que, dans le cas particulier des expressions de concaténation, les grandes combinaisons peuvent très souvent être séparées en d'autres plus courtes. Ce découpage peut être réalisé en regroupant les concepts adjacents dont les noms partagent un même préfixe ou suffixe. Par exemple, l'expression *concat(street1, street2, zip, city, state, country)* peut être scindée en deux sous-combinaisons soient *concat(street1, street2)* et *concat(zip, city, state, country)*. Le but de cette opération de division est de générer un grand nombre de concepts complexes cohésifs qui seraient autrement éliminés par le second filtre. Le tableau 8.4 montre les résultats de nos expériences relativement à la découverte des concepts complexes par analyse des artefacts constituant le contexte des sources de données. Les seuils liés aux deux filtres ont été respectivement fixés à 0.8 (similarité) et 5 (nombre de concepts). Avant l'application des filtres, 277 concepts complexes ont initialement été extraits à partir du contexte de *PetStore* qui comptait 1123 documents. En revanche, 48 concepts complexes ont été découverts dans l'application *eStore* dont le contexte comptait seulement 72 documents. Ceci constitue un résultat intéressant en soi puisqu'il suggère que l'efficacité du repérage de concepts complexes dépend essentiellement, non pas de la quantité, mais de la pertinence des artéfacts composant le contexte d'une source de données. Tel qu'indiqué dans le Tableau 8.4, lorsque les filtres ont été appliqués, l'*analyseur de contexte* a pu découvrir un total de 82 concepts complexes dont 79% étaient cohésifs. Quant aux concepts complexes pertinents, 12 ont été découverts sur un maximum de 15, pour une valeur globale de pertinence de 80%. Il est important de souligner que notre *analyseur de contexte* n'a pas été développé en fonction des applications choisies pour notre étude de cas. La généricité et l'extensibilité sont des qualités intrinsèques à l'architecture de l'*analyseur de contexte*. En effet, les modules de l'analyseur sont conçus pour supporter une «catégorie» générale de documents plutôt qu'un format spécifique de fichier. Par exemple, l'implémentation propose un seul *analyseur de programmes* traitant tous les programmes C#, Java et coldfusion. À partir d'une conìguration exploitant à

TAB. 8.5 – Résultats de performance du mapping

	matching global						matching complexe		
	$préc_1$	rap_1	$f\text{-}m_1$	$préc_2$	rap_2	$f\text{-}m_2$	préc	rap	f-m
PetStore/eStore	70%	56%	62%	87%	49%	63%	100%	100%	100%
PetStore/PetShop	67%	62%	64%	68%	66%	67%	100%	40%	57%
PetStore/PetMarket	54%	84%	66%	55%	86%	67%	100%	50%	67%
eStore/PetShop	83%	64%	73%	87%	68%	77%	100%	100%	100%
eStore/PetMarket	67%	69%	68%	68%	70%	69%	100%	100%	100%
PetShop/PetMarket	72%	79%	75%	76%	79%	78%	100%	100%	100%
Average	69%	69%	69%	73%	69%	70%	100%	82%	**87%**

N.b. **préc** : précision ; **rap** : rappel ; **f-m** : f-mesure ; $_1$: sans enrichissement i.e. matching simple seulement ; $_2$: avec enrichissement i.e. matching simple et complexe ;

peine cinq règles d'extraction heuristiques, l'*analyseur de contexte* réalise déjà une bonne performance comme le démontrent les résultats du tableau 8.4.

8.3.2 Évaluation du matching complexe

Les expériences réalisées pour évaluer la performance du matching complexe ont reposé sur deux types d'exécutions : soient des exécutions avec enrichissement des sources de données par des concepts complexes, et d'autres sans enrichissement. Dans le cas d'une exécution avec enrichissement, seuls les concepts complexes cohésifs ont été utilisés. La performance liée à chaque alignement généré est évaluée en utilisant la *f-mesure*. Le tableau 8.5 montre que le *module aligneur* a bien performé pour le matching complexe puisqu'il présente une moyenne de *f-measure* de 87%. Nous pouvons aussi constater que l'enrichissement des sources de données par les concepts complexes n'a pas eu d'effet négatif sur la performance du matching simple. En effet, on constate que $préc_1 \leq préc_2$ et donc qu'il n'y a pas eu de perte due aux effets de bruit que l'on pouvait craindre.

8.4 Expérimentation du mapping dynamique

Les expériences menées relativement au mapping dynamique ont elles aussi visé l'alignement des sources de données associées aux quatre applications de vente d'animaux en ligne : *Java Pet Store* [Mic05], *eStore* [McU03], *PetMarket* [Ado07] et *PetShopDNG* [Dot03]. Notre principal objectif de test était d'évaluer l'impact de la com-

binaison dynamique des résultats de matching des stratégies individuelles d'alignement par les coordonnateurs sur la performance globale d'INDIGO. Ainsi, dans une première série de tests, nous avons comparé l'efficacité liée à l'utilisation d'une combinaison basée sur le stacking avec celle utilisant la moyenne. Ensuite, dans une seconde série des tests nous avons mis à l'épreuve INDIGO face aux systèmes SF, V-doc et Cupid en dotant son module aligneur de la capacité du mapping dynamique.

L'architecture de INDIGO que nous avons utilisée dans les tests est illustrée par la figure 8.6. En plus du superviseur, elle comporte deux aligneurs. Un aperçu de chaque module est donné ci-après.

- *Aligneur de noms.* L'aligneur de noms est basé sur la métrique de similarité lexicale *Q-Gram* [EBB+04]. Il normalise les noms de concepts avant de procéder à leur alignement.
- *Aligneur de contenus.* L'aligneur de contenus utilise une stratégie de matching basée sur une version adaptée de la technique Whirl développée par Cohen et Hirsh [CH98] (cf. chapitre 6)

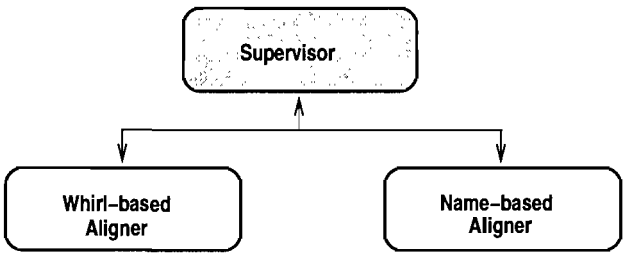


FIG. 8.6 – Architecture du module aligneur configurée pour évaluer le mapping dynamique

8.4.1 Mesure de l'efficacité du stacking

Les tests ont été menés en deux étapes. (1) Dans une première étape, le module superviseur a été configuré pour combiner les résultats retournés par ses modules subordonnés en utilisant la moyenne. INDIGO a ensuite été exécuté pour aligner six paires de sources de données. (2) Dans la seconde étape, le module superviseur a été configuré pour générer le mapping final en combinant les matchings individuels par la technique

de stacking. Le processus d'alignement basé sur cette technique a été exécuté en deux phases :

1. *Phase d'entraînement* : Nous avons commencé par définir les descripteurs de concepts qui sont susceptibles d'influencer la performance des deux aligneurs composant l'architecture d'INDIGO (c.f. figure 8.6). Nous avons alors choisi le *status d'abréviation* comme la propriété influençant l'efficacité de l'aligneur de noms. Tandis que pour l'aligneur de contenus, nous avons opté pour un descripteur lié au type des données d'un concept. En effet, l'expérience [DDH03] a montré que l'aligneur de contenus est particulièrement efficace quand il s'agit d'aligner des concepts de type textuel (surtout descriptif). Par conséquent, nous avons décidé d'utiliser le *Status de type de données* comme le second descripteur composant la signature des concepts (cf. chapitre 7 pour plus de détails sur la notion de signature et son utilisation dans un processus de stacking). Ce descripteur prend la valeur 1 si le concept correspondant est de type textuel et prend la valeur 0 dans le cas contraire. L'ensemble global des classes s'est alors composé des quatre signatures de concepts suivantes : (0,0), (0,1), (1,0) et (1,1). Par ailleurs, afin d'assurer que les résultats d'alignement ne soient pas biaisés par cette phase d'entraînement, nous avons opté pour une approche basée sur la technique de validation croisée. En effet, étant donné une paire de sources de données à aligner, les données d'entraînement ont été constituées à partir des mappings de références relatifs aux cinq autres paires de sources de données.
2. *Phase d'alignement* : Nous avons exécuté le module aligneur pour réconcilier chacune des six paires de sources de données. À chaque exécution, le module superviseur a utilisé les poids estimés durant la phase d'entraînement pour combiner les résultats de matching renvoyés par ses modules subordonnés.

Pour faire le suivi des variations de performance d'INDIGO, la *f-mesure* a encore une fois été utilisée. Le tableau 8.6 regroupe les résultats obtenus lors de notre première série de tests. Ces résultats montrent que la technique de stacking a un impact positif sur la performance du module aligneur d'INDIGO. En effet, grâce à l'utilisation de cette

	combinaison basée sur la moyenne			combinaison basée sur le stacking			Variation		
	prec.	*rec.*	*f-m.*	*prec.*	*rec.*	*f-m.*	*prec.*	*rec.*	*f-m.*
eStore/petStore	72.73	53.33	61.54	81.82	60.00	69.23	+12%	+13%	+12%
petShop/eStore	62.50	50.00	55.56	42.86	60.00	50.00	−31%	+20%	−10%
petShop/petStore	54.55	50.00	52.17	61.54	66.67	64.00	+13%	+33%	+22%
petMarket/petShop	71.43	41.67	52.63	66.67	50.00	57.14	−7%	+20%	+8%
petMarket/eStore	30.00	20.00	24.00	42.86	40.00	41.38	+43%	+100%	+72%
petMarket/petStore	33.33	38.46	35.71	57.14	30.77	40.00	+71%	−20%	+12%
Average	54.09	42.24	46.93	58.81	51.24	53.62	+9%	+21%	**+15%**

TAB. 8.6 – Variation de performance due à l'application de la technique de stacking

technique la *f-mesure* a pu être améliorée de 15% globalement. Dans le cas particulier de l'étude de cas petMarket/eStore, la *f-mesure* du processus d'alignement a même pu être augmentée de 72%. En revanche, dans le cas spécifique de l'alignement de la paire de sources de données petShop/eStore, la performance a plutôt diminué de 10%. Ceci montre le côté faible d'une technique d'apprentissage, telle que celle du stacking, qui dépend fondamentalement de la qualité des données utilisées pour l'entraînement. Néanmoins, dans tous les autres cas, elle a permis d'améliorer la performance d'INDIGO.

8.4.2 Comparaison avec d'autres systèmes de mapping

INDIGO a encore une fois été mis à l'épreuve face aux systèmes SF, V-Doc et Cupid. Le tableau 8.7 décrit les résultats de test réalisés sur ces systèmes. Le tableau 8.8 montre leur performance globale à côté de celle d'INDIGO. Les deux dernières colonnes présentent les résultats réalisés par INDIGO pour (1) l'alignement basé sur la moyenne et (2) celui utilisant la technique de stacking.

Globalement, INDIGO a réussi à dépasser tous les autres systèmes. En effet, la performance d'INDIGO a pu être améliorée de 15% grâce à l'utilisation de la technique de stacking, ce qui lui a permis de se positionner au premier rang dans le classement de ces quatre systèmes après qu'il y occupait la seconde place.

Enfin, il est important de remarquer l'absence de biais dans nos expériences. En effet, nos données de tests se sont constituées de quatre applications dans le domaine du commerce électronique et de deux ontologies de cours d'universités américaines qui

	V-Doc			SF			Cupid		
	prec.	*rec.*	*f-m.*	*prec.*	*rec.*	*f-m.*	*prec.*	*rec.*	*f-m.*
eStore/petStore	100,00	26.66	42.10	80.00	36.36	50.00	80.00	26.67	40.00
petShop/eStore	100.00	10.00	18.18	40.00	20.00	26.66	27.27	30.00	28.57
petShop/petStore	100.00	58.33	73.68	85.71	60.00	70.58	87.50	58.33	70.00
petMarket/petShop	100.00	25.00	40.00	80.00	33.33	47.00	66.67	33.33	44.44
petMarket/eStore	33.33	6.67	11.11	83.33	41.66	55.55	30.77	26.67	28.57
petMarket/petStore	100.00	30.76	47.05	50.00	38.46	43.47	61.54	61.54	61.54
Average	88.89	26.24	38.69	69.84	38.30	48.88	58.96	39.42	45.52

TAB. 8.7 – Résultats de matching des systèmes SF, V-Doc et Cupid

	V-Doc	SF	Cupid	INDIGO(1)	INDIGO(2)
précision	88.89	69.84	58.96	54.09	58.81
rappel	26.24	38.30	39.42	42.24	51.24
f-mesure	38.69	48.88	45.52	46.93	53.62

TAB. 8.8 – Comparaison de la performance globale de INDIGO avec les résultats de matching des systèmes V-Doc, SF et Cupid

ont été développées par d'autres tiers et que nous avons récupérées à partir d'Internet puis directement utilisées dans notre expérimentation sans les modifier. Par ailleurs, nous avons tâcher à tout moment de configurer INDIGO par des stratégies d'alignement déjà utilisées par d'autres systèmes cités en littérature (e.g. la technique Whirl [CH98] et métrique de matching lexical JARO). De cette façon, nous avons évité le risque de développer une stratégie de matching qui soit efficace pour une étude de cas spécifique. Aussi, ceci nous a permis de confirmer le fait que la contextualisation peut améliorer les techniques de matching déjà existantes et que le stacking peut conférer à une architecture multi-stratégique la capacité d'adaptativité indépendamment du choix des stratégies qui la composent.

8.5 Conclusions

Ce chapitre présente les expériences que nous avons menées pour valider le bien fondé et l'efficacité de l'approche que nous avons présentée dans le chapitre 5. Ces expériences ont permis de valider nos trois contributions majeures dans le cadre de la présente thèse. Elle nous ont ainsi permis de démontrer que les solutions suggérées (cf. chapitres 6, 7

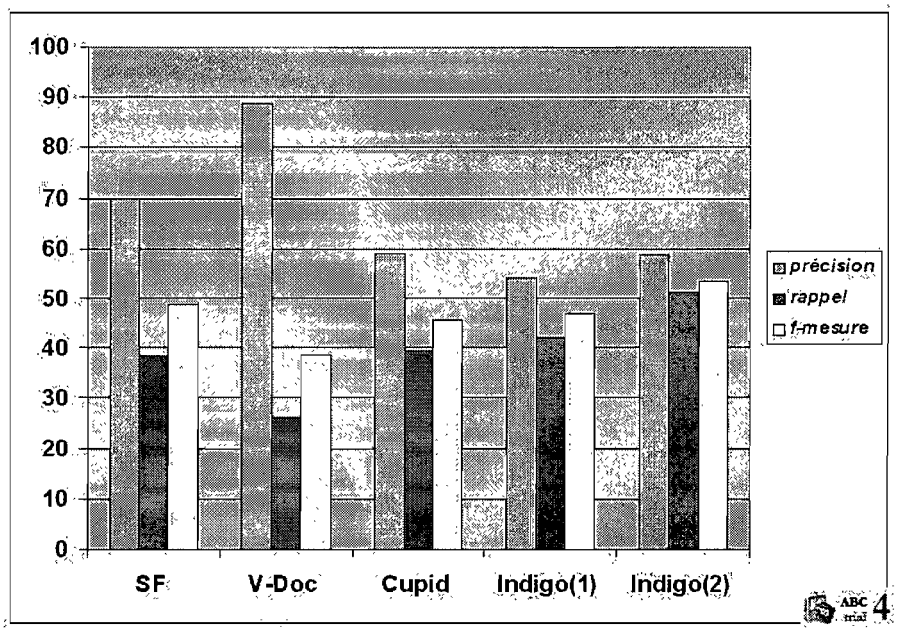


FIG. 8.7 – Graphique illustrant les résultats de Comparaison de la performance globale de INDIGO avec les résultats de matching des systèmes V-Doc, SF et Cupid

et 8) aident effectivement à mettre au point un système d'alignement sémantique des données performant et compétitif par rapport aux autres systèmes existants.

CHAPITRE 9

CONCLUSION

Le besoin d'interopérabilité sur Internet est devenu de nos jours très imminent vu le nombre important d'acteurs écomomiques qui l'adoptent comme moyen standard de communication. Nous assistons d'ailleurs à la création d'un nombre croissant de places de marché et de consortium virtuels dont les exigences importantes en interopérabilité sont bien connues. Nous retrouvons ce genre d'organisations dans divers domaines économiques tels que l'industrie de voyage, l'industrie chimique ou pharmaceutique, etc. Toutefois, puisqu'elle implique l'intégration des données des membres d'un certain groupement d'entités commerciales, l'interopérabilité se retrouve confrontée à un obstacle majeur qui est celui de l'hétérogénéité des sources de données. En effet, pour que cette intégration soit possible, elle doit nécessairement passer par l'alignement sémantique entre les données sources et cibles (i.e. mise en correspondance entre concepts partageant une sémantique commune). Il s'est avéré qu'en pratique cette étape est très laborieuse et sujette à de nombreuses erreurs si elle est effectuée manuellement. Le présent travail de recherche expose les aspects importants liés à la problématique d'alignement sémantique des données et propose une nouvelle plateforme appelée INDIGO dont l'architecture, extensible et adaptative, se veut une solution efficace pour un mapping semi-automatique entre sources de données hétérogènes. La section suivante rappelle les contributions majeures que nous avons réalisées dans le cadre de cette recherche. Elle est suivie par une seconde section présentant les perspectives futures relatives au développement de notre système actuel INDIGO.

9.1 Contributions clés

Le principal enjeu de l'alignement sémantique automatique est bien sûr sa performance tant du point de vue de la précision que du rappel. Dans cette perspective, la présente thèse a proposé, implémenté et validé deux solutions novatrices au problème.

Ces deux contributions se résument comme suit :

1) **Intégration du contexte externe des sources de données à aligner :** Traditionnellement, seuls les schémas de sources de données et leurs instances sont utilisées comme gisement d'informations pour la caractérisation sémantique des concepts à aligner. Nous avons montré dans cette thèse qu'il était possible d'enrichir ces informations déjà présentes en fouillant dans les documents composant le contexte des sources de données. Cette approche présente l'avantage majeur d'exploiter le cadre de conceptualisation des sources de données qui est en principe riche en informations cognitives cachées derrière ces sources. Il se compose des différents artéfacts produits habituellement dans un processus de développement ou de maintenance d'une application et qui appartiennent à l'environnement externe d'une source de données (dossiers de conception, programmes, formulaires, dictionnaires de données, etc.). Nous distinguons deux types de contextes : le contexte opérationnel et le contexte descriptif. Le contexte opérationnel est composé des ressources respectant des grammaires syntaxiques prédéfinies (e.g. programmes informatiques, OCL, XMI, etc.), tandis que le contexte descriptif est constitué des ressources décrites dans un format libre (e.g. spécification des besoins, dossiers de conception, etc.). Nous avons conçu un module appelé *analyseur de contexte* pour l'exploration systématique des contextes opérationnel et descriptif en vue d'en extraire des informations sémantiques destinées à enrichir les sources données correspondantes. Deux types d'enrichissements ont été pris en charge :

   • Extension par des concepts complexes (e.g. *(1+taxe)*montant*, *concat(rue, ville, code_postal)*) en vue de la découverte de mappings complexes (1 à n, n à 1 et n à m). Les concepts complexes peuvent notamment être détectés à partir de formules présentes dans le contexte opérationnel (e.g. dans une requête SQL).

   • Association aux noms des concepts, des mots se trouvant fréquemment dans leur voisinage au niveau du contexte descriptif.

   Pour assurer une analyse maîtrisée et efficace du contexte, nous avons proposé une architecture hiérarchique pour l'analyseur de contexte basée sur de multiples

analyseurs spécialisés et supervisés par des méta-analyseurs. Cette architecture a donc l'avantage d'être facilement adaptable et extensible.

2) Implémentation d'un système d'alignement multi-stratégique et adaptatif : Nous avons implémenté un module aligneur travaillant de concert avec l'analyseur de contexte. Une fois enrichies par ce dernier, les sources de données sont prises en charge par le module aligneur pour leur mapping effectif. L'architecture du module aligneur est aussi composée de plusieurs modules organisés hiérarchiquement. Les modules occupant les feuilles au sein de la hiérarchie implémentent chacun une stratégie d'alignement donnée qui exploite un type d'information sémantique spécifique (e.g. nom d'un concept, type de données, etc.). Il sont appelés aligneurs. Les modules représentant des noeuds, quant à eux, sont appelés coordonnateurs. Ils ont pour rôle de superviser les actions de leurs modules subordonnés (i.e. aligneurs et/ou coordonnateurs) et de combiner leurs résultats de matching respectifs. Chaque coordonnateur se distingue par sa capacité de sélectionner dynamiquement une stratégie individuelle d'alignement à appliquer parmi plusieurs pour le traitement d'un type d'information sémantique donné (e.g. nom d'un concept, type de données, etc.). Le choix de la meilleure stratégie dépendra des caractéristiques sémantiques du concept cible à aligner.

L'utilisation d'INDIGO pour l'alignement entre sources de données issues du domaine du commerce éléctronique a confirmé la pertinence de notre approche. En effet, concernant le matching lexical, nous avons démontré que la contextualisation de sources de données peut améliorer considérablement la performance d'un processus d'alignement. Par ailleurs, nos expériences ont aussi montré qu'INDIGO pouvait effectivement offrir de bonnes performances en ce qui concerne l'extraction de concepts complexes et pouvait générer efficacement un matching complexe. Enfin, nos tests ont montré que le mapping adaptatif permet effectivement d'améliorer la performance du processus d'alignement. En particulier, INDIGO a été comparé avec trois systèmes de matching connus dans le domaine et les a tous dépassés.

9.2 Perspectives futures

Considérant sa performance, INDIGO ouvre la voie à de nouvelles solutions adaptatives exploitant le contexte de sources pour faire face à la diversité des données sur le Web et faciliter la réconciliation de leur sémantique. En effet, la majorité des sources de données sur le Web sont documentées. Par exemple, un catalogue de vente en ligne de produits dépend d'une application de base de données documentée dans l'arrière-guichet et est par conséquent inévitablement liée à un ensemble d'artefacts. De façon similaire, les ontologies dans le domaine du commerce électronique constituent en soit des modélisations de domaines spécifiques bien documentés. Nous croyons donc que la contextualisation des données constitue une solution réaliste et pragmatique pour renforcer la sémantique des données sur le web. Parmi les idées intéressantes d'application, citons l'exemple d'enrichissement des registres UDDI par des informations contextuelles facilitant l'identification des services et des produits. En effet, les codes des services et produits listés sur les pages jaunes pourraient être enrichis par des noms de concepts extraits à partir de la documentation créée par les entreprises qui les publient. Les solutions d'alignements basées sur le contexte, comme celle présentée dans la présente thèse, pourraient donc certainement contribuer à l'identification efficace de ces services et produits.

Par ailleurs, nous comptons miser sur l'extensibilité du module analyseur de contexte d'INDIGO pour ajouter de nouveaux types d'enrichissement de sources de données. En particulier, nous prévoyons étendre l'enrichissement lexical (qui se fait actuellement par simple association de mots extraits du contexte avec les concepts d'une source de données) en générant des graphes établissant des relations sémantiques entre ces mots et concepts. Par exemple, on pourrait créer la relation *acheter* entre le mot d'enrichissement **client** et le concept de la source de données **produit**. Nous pensons que cela offrirait au module aligneur une source d'informations plus riche sémantiquement et qu'il pourrait exploiter pour améliorer la performance globale du matching lexical.

Enfin, nous projetons de prendre en charge l'identification et le matching de concepts complexes pour répondre aux besoins d'alignement particuliers des bases de données

orientées objet. Nous comptons notamment exploiter la logique présente dans les bases objets, de même que la notion d'héritage et les requêtes *OQL*.

BIBLIOGRAPHIE

[ACC04] The accord project. University of Trento (Italy), Web site, 2004. `http://www.dit.unitn.it/~accord/`.

[Ado07] Adobe. Petmarket. Web Site, 2007.

[BC86] J. Biskup and B. Convent. A formal view integration method. In *In Proceedings of the ACM Conf. on Management of Data (SIGMOD)*, 1986.

[BLG04] R. Dieng-Kuntz B.T. Le and F. Gandon. On ontology matching problems - for building a corporate semantic web in a multi-communities organization. In *Proc. of the Int. Conf. on Enterprise Information Systems (ICEIS)*, volume 4, pages 236–243, 2004.

[BV07] I.Y. Bououlid and J. Vachon. A context-based approach for complex semantic matching. In *Proc. of CAiSE-Forum*, Trondheim, Norway, 2007.

[CH98] W. Cohen and H. Hirsh. Joins that generalize : Text classification using whirl. In *Proc. International Conference on Knowledge Discovery and Data Mining (KDD'98)*, 1998.

[DAR05] S. Massmann D. Aumueller, H.H. Do and E. Rahm. Schema and ontology matching with coma++. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 906–908, 2005.

[DDH03] A. Doan, P. Domingos, and A. Halevy. Learning to match the schemas of data sources : A multistrategy approach. *Journal of Machine Learning*, 50(3) :279–301, 2003.

[DH73] R.O. Duda and P.E. Hart. Pattern classification and scene analysis. 1973.

[Doa02] A. Doan. Learning to map between structured representations of data, 2002. PhD thesis.

[Dot03] DotNetGuru.org. Petshopdng. Web Site, 2003.

[DR02] Hong-Hai Do and Erhard Rahm. Coma : A system for flexible combination of schema matching approaches. In *Proceedings of the 28th Conf. on Very Large Databases (VLDB)*, 2002.

[EBB+04] J. Euzenat, T.L. Bach, J. Barrasa, P. Bouquet, J. De Bo, R. Dieng, M. Ehrig, M. Hauswirth, M. Jarrar, R. Lara, D. Maynard, A. Napoli, G. Stamou, H. Stuckenschmidt, P. Shvaiko, S. Tessaris, S.V. Acker, and I. Zaihrayeu. State of the art on ontology alignment, AUG 2004. Part of a research project funded by the IST Programme of the Commission of the European Communities as project number IST-2004-507482.

[Gro06] XObjects Group. V-doc system. Web site, 2006. `http://xobjects.seu.edu.cn/project/falcon/res/`.

[HCZ+06] W. Hu, G. Cheng, D. Zheng, X. Zhong, and Y. Qu. The results of falcon-ao in the oaei 2006 campaign. In *International Workshop on Ontology Matching*, 2006.

[He04] Chang K. C.-C. Han. J. He, B. Discovering complex matchings across web query interfaces : A correlation mining approach. In *Proceedings of the SIGKDD conf.*, 2004.

[Jar95] M. Jaro. Probabilistic linkage of large public health data files. In *Statistics in Medicine*, volume 14, pages 491–498, 1995.

[JEV05] P. Guegan J. Euzenat and P. Valtchev. Ola in the oaei 2005 alignment contest. In *Workshop on Integrating Ontologies 2005*, pages 97–102, 2005.

[JPE01] J.Madhavan, P.Bernstein, and E.Rahm. Generic schema matching using cupid. In *Proceedings of the 27th VLDB Conference, Roma, Italy (VLDB)*, pages 48–58, 2001.

[KKS04] G.A. Vouros K. Kotis and K. Stergiou. Capturing semantics towards automatic coordination of domain ontologies. In *Artificial Intelligence : Methodology, Systems, and Applications*, volume 3192 of *LNCS*, pages 22–32, 2004.

[LC94] Wen-Syan Li and Chris Clifton. Semantic integration in heterogeneous databases using neural networks. pages 1–12, 1994.

[Len02] M. Lenzerini. Data integration : A theoretical perspective. In *Proceedings of PODS*, pages 233–246, 2002.

[Lev66] I. Levenstein. Binary codes capable of correcting deletions, insertions and reversals. *Sov. Phys. - Dokl*, 10(8) :707–710, 1966.

[lpg07] Natural language processing group. Gate. Web site, 2007. `http://www.dcs.shef.ac.uk/nlp/gate/`.

[McU03] R. McUmber. Developing pet store using rup and xde. Web Site, 2003.

[Mel03] S. Melnik. Similarity flooding. Web site, 2003. `http://infolab.stanford.edu/~melnik/mm/sfa/`.

[MGMR02] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding : a versatile graph matching algorithm and its application to schema matching. In *Proceedings of the 18th International Conference on Data Engineering (ICDE)*, San Jose (CA US), 2002.

[MHDB02] J. Madhavan, A. Halevy, P. Domingos, and P. Bernstein. Representing and reasoning about mappings between domain models. In *In Proceedings of the National AI Conference (AAAI-02)*, 2002.

[Mic05] Sun Microsystems. Java petstore. Web Site, 2005.

[MS99] C. Manning and H. SchÄutze. Foundations of statistical natural language processing. 1999.

[NHUTO92] H. Ney, R. Haeb-Umbach, B.-H. Tran, and M. Oerder. Improvements in beam search for 10000-word continuous speech recognition. In *Proc. International Conference on Acoustics, Speech, and Signal Processing*, 1992.

[NM01] N. Noy and M. Musen. Anchor-prompt : Using non-local context for semantic matching. In *In Proc. IJCAI 2001 workshop on ontology and information sharing*, pages 63–70, Seattle (WA US), 2001.

[Noy04] N.F. Noy. Semantic integration : A survey of ontology-based approaches. *SIGMOD Record, Special Issue on Semantic Integration*, 33(4), 2004.

[NW70] S.B. Needleman and C.D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. In *Molecular Biology*, volume 48, pages 444–453, 1970.

[QHC06] Y. Qu, W. Hu, and G. Cheng. Constructing virtual documents for ontology matching. In *Proceedings of the 15th International World Wide Web Conference*, 2006.

[RLD+04] R.Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos. imap : Discovering complex semantic matches between database schemas. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 383–394, 2004.

[RW86] V.V. Raghavan and S.K.M. Wong. *A Critical Analysis of Vector Space Model for Information Retrieval*, volume 37(5). JASIS, 1986.

[SM83] G. Salton and M.H. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, 1983.

[ST95] E. Sutinen and J. Tarhio. On using q-gram locations in approximate string matching. In *Proceedings of the Third Annual European Symposium on Algorithms*, page 327 340. Springer Verlag, 1995.

[SW81] T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. In *Journal of Molecular Biology*, volume 147, page 195 197, 1981.

[TW99] K. M. Ting and I. H. Witten. Issues in stacked generalization. *Journal of Artificial Intelligence Research*, 10 :271–289, 1999.

[W3C07] W3C. Semantic web. Web site, 2007.

[Win99] W. Winkler. The state record linkage and current research problems. In *Technical report, Statistics of Income Division, Internal Revenue Service Publication*, 1999.

[Wol92] D. Wolpert. Stacked generalization. *Neural Networks*, pages 241–259, 1992.

[XE03] L. Xu and D. Embley. Using domain ontologies to discover direct and indirect matches for schema elements. In *Proceedings of the Semantic Integration Workshop*, 2003.