Mise en correspondance de schémas conceptuels

Ibrahima KHOULE, Mohamed BENJELLOUN

Février 2013

Résumé :

Dans le domaine de l'ingénierie logicielle, l'intégration de différentes sources de données devient de plus en plus récurrente, en effet, beaucoup de projets sont réalisés en utilisant des sources de données externes, qu'il faut donc intégrer. Ce mécanisme peut être automatisé de manière à simplifier l'étude de données lors d'une conception logicielle. Ce document présente un algorithme permettant d'aligner les différents éléments de deux modèles de données représentant le même concept sémantique ; ce dernier est basé sur une étude détaillée de différentes approches existant dans ce domaine. Une étape de validation et d'expérimentation a été ensuite effectuée sur différents schémas conceptuels à l'aide du prototype développé.

Mots-clés : mise en correspondance de modèles, alignement de modèles, schéma conceptuel.

1 – Introduction

1.1 – Contexte

Ce TER s'inscrit dans le cadre de l'UE Mif20 « Projet de recherche », et porte sur la thématique de conception d'applications basées sur différentes sources de données déjà existantes. La conception d'une application logicielle passe par une étude des exigences fonctionnelles pour définir le modèle logique de données qu'il sera alors nécessaire d'implanter. Ces applications peuvent également au moment de leur conception reposer sur des données issues d'autres sources distantes. Dans le contexte de ce projet, nous supposons disposer de modèles conceptuels de données, que l'on peut obtenir par rétro-ingénierie [JD13].

L'objectif de cette étude est de permettre d'identifier automatiquement les correspondances entre différents modèles conceptuels (cf. Figure 1), en se basant sur des approches existantes pour l'alignement des modèles logiques et en l'adaptant à l'alignement des modèles conceptuels.

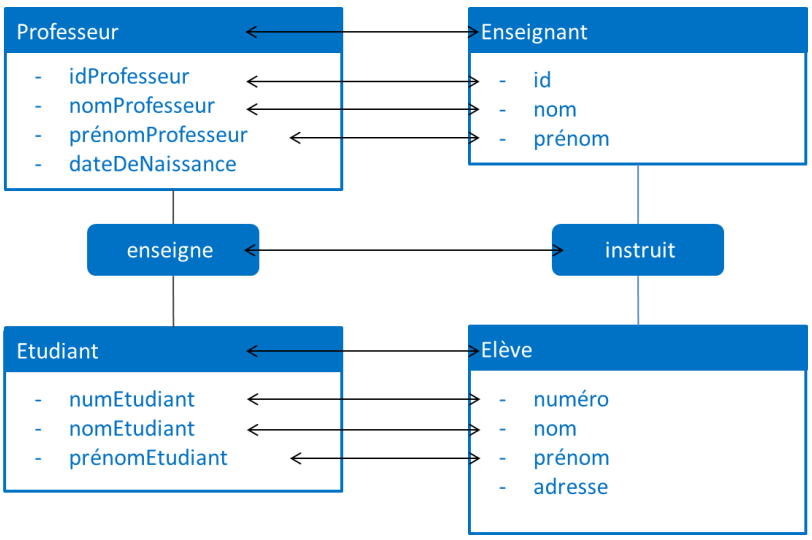


Figure 1 : Exemple de deux modèles conceptuels et leur alignement (une flèche représente une correspondance)

Un prototype nommé *« Database Matching »* et implémentant la solution proposée par cette étude a été mis en œuvre, ce prototype a pour vocation, à moyen terme, d'être intégré dans une plateforme de conception de bases de données actuellement en cours de développement au sein de l'équipe BD de LIRIS (Laboratoire d'Informatique en Image et Systèmes d'information).

1.2 – Problématique

La nécessité d'aligner deux schémas conceptuels est née du besoin d'intégrer des bases de données hétérogènes, développées indépendamment les unes des autres et dont les modèles ne partagent pas le même vocabulaire. Les différentes solutions étudiées dans le domaine de la mise en correspondance des modèles ne sont pas parfaitement adaptées aux schémas entité/association qui représentent notre cas d'étude. En effet, les algorithmes existants n'exploitent pas les différentes relations existantes entre les attributs, les associations et les entités lors de l'alignement, or ces dernières contribuent à définir la sémantique des différents éléments à aligner. Par exemple dans la Figure 1, le fait de savoir que « nomEtudiant » appartient à un « Etudiant », et « nom » appartient à un « Elève » nous permet de faire correspondre ces deux attributs, car l'entité représente le contexte auquel l'attribut est défini.

Le but de ce travail de recherche consiste donc à détecter les points forts des approches existantes, et les adapter pour aligner des schémas entité/association. Une seconde contribution réalisée lors de ce projet est la définition d'une mesure de similarité basée sur les préfixes. Enfin, nous avons validé notre approche sur des jeux de données utilisées en cours de Bases de Données.

2 – Etat de l'art

Les solutions existantes dans le domaine de la mise en correspondance de modèles de données proposent différentes approches pour traiter le problème.

2.1 – Algorithme de « Similarity Flooding »

Dans l'article [MGR02], l'algorithme d'inondation par similarité (ou « Similarity Flooding ») est défini en cinq étapes, il prend en entrée deux graphes orientés étiquetés correspondant aux modèles à aligner, et fourni un ensemble d'alignements entre ces graphes.

La première étape de l'algorithme consiste à générer un graphe représentant toutes les compatibilités possibles entre les éléments des deux graphes à aligner ; cela s'effectue de la manière suivante : Pour chaque triplet *(S, r, T)* du premier graphe, où *S* et *T* sont respectivement le nœud source et le nœud destination, et r la relation entre *S* et *T*, s'il existe un triplet *(U, r, V)* dans le deuxième graphe (*r* étant la même relation que dans le premier triplet), les deux couples *(S, U)* et *(T, V)* appelés « *nœuds de compatibilité* » sont alors ajoutés au graphe de compatibilité, et sont reliés par la relation *r* (cf. Figure 2).

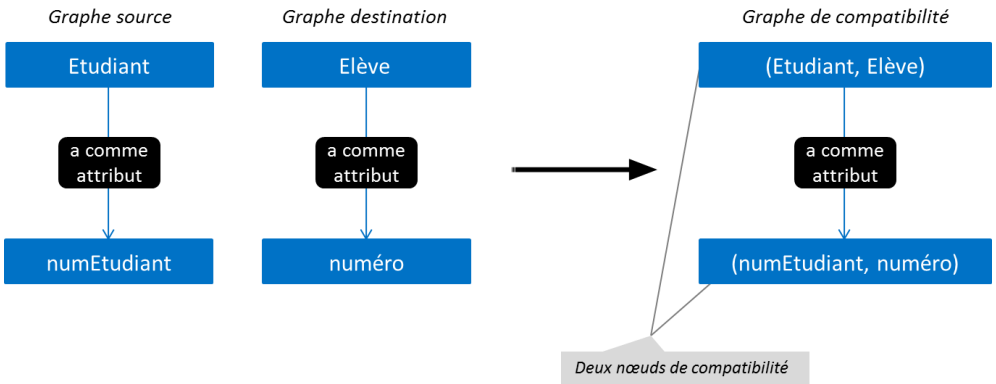


Figure 2 : Création de graphe de compatibilité

La deuxième étape de l'algorithme consiste à calculer un graphe de propagation en utilisant le graphe de compatibilité ; il s'effectue comme suit :

- Pour chaque arc *(S, r, T)* du graphe de compatibilité (*S* et *T* sont deux nœuds de compatibilité, et *r* est la relation entre ces nœuds), un arc inverse *(T, r, S)* est créé.
- Chaque triplet *(S, r, T)* du nouveau graphe est remplacé par *(S, 1/x, T)* où x est le nombre d'arcs sortant du nœud de compatibilité S.

La troisième étape de l'algorithme consiste à assigner des valeurs de similarité initiales à chaque nœud de compatibilité. Une formule mathématique basée sur *la distance de Levenshtein*[1] représente la valeur de similarité initiale de chaque nœud de compatibilité. La quatrième étape consiste à calculer un point-fixe pour chaque nœud de compatibilité, à l'aide d'une formule mathématique réitérée jusqu'à l'obtention d'une valeur de similarité stable (qui ne change pas entre deux itérations successives). La dernière étape consiste simplement à fixer une valeur de seuil, et filtrer tous les nœuds de compatibilité pour n'en garder que ceux dont la valeur de similarité est supérieure ou égale à ce seuil.

2.2 – Approche « GUMM »

Dans l'article [FHLC08], les auteurs proposent une approche basée sur le Similarity Flooding pour aligner deux modèles de données passés en entrée (cf. Figure 3). Plus précisément, l'approche consiste à transformer chacun des modèles d'entrée en un graphe orienté étiqueté ; on applique ensuite l'algorithme de *Similarity Flooding* à ces deux graphes ce qui permet de découvrir les paires d'éléments possibles à aligner, et la dernière étape consiste à générer un modèle intégré à partir de cet alignement.

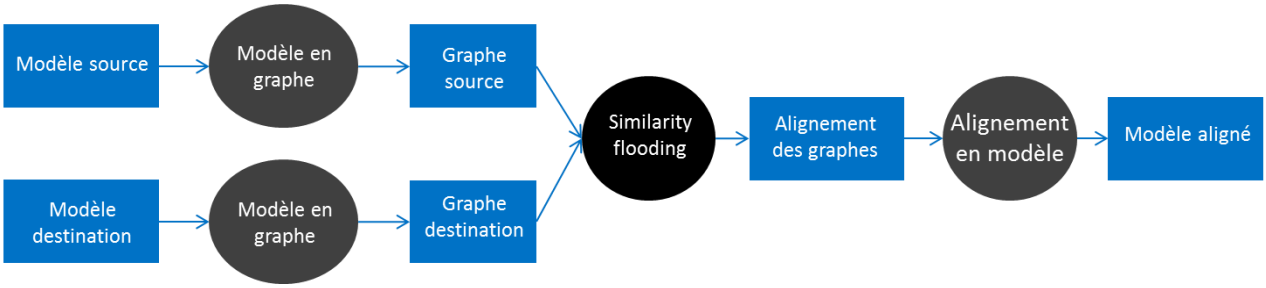


Figure 3 : Schéma représentant l'approche GUMM

2.3 – Approche « MT4MDE »

Cette approche, comme le détaille l'article [LHSB06], permet de générer un alignement automatique (MT4MDE) ou semi-automatique (SAMT4MDE) entre deux modèles passés en entrée. Plusieurs mesures de similarité sont prises en considération dans cette approche et aident à déterminer le niveau de similarité entre les éléments source et les éléments destination, ces mesures sont séparées en plusieurs niveaux, et peuvent être employées individuellement ou en combinaison d'autres mesures afin de produire un alignement :

- « *Element level* » se basant sur les contraintes d'intégrité (ex. les types des différents éléments à aligner), et la similarité terminologique (synonymie et description des éléments à aligner).
- « *Structure level* » repose sur la relation entre les différents éléments à aligner, ce niveau nécessite donc la création d'un graphe d'alignement et se base uniquement sur les contraintes d'intégrité.
- « *Instance/Contents-based level* » est basé sur la fréquence d'apparition des termes dans les éléments à aligner ainsi que sur le contexte dans lequel ces éléments sont définis.

Dans tous les cas, l'approche exige l'intervention d'un expert qui accepte ou refuse les différentes correspondances trouvées par l'algorithme (section IV, [LHSB06]), auquel cas l'expert peut ajouter, supprimer ou modifier certaines correspondances proposées. Ces modifications peuvent ensuite être utilisées pour calculer la qualité des mesures de similarité choisies.

[1] : La distance de Levenshtein est une mesure de similarité entre deux chaînes de caractères, égale au nombre de caractères qu'il faut supprimer, insérer ou modifier pour passer d'une chaîne à l'autre.

2.4 – Positionnement

Les différentes solutions étudiées ne répondent pas parfaitement à nos attentes, car ils omettent plusieurs critères pouvant être pris en considération lors d'une mise en correspondance de schémas entité/association, notamment les relations entre les différents éléments d'un schéma (entité-attribut, entité-association, entité-entité etc.). La combinaison des bonnes idées de l'approche « GUMM » avec celles de « MT4MDE » pourrait donner un algorithme adapté à notre besoin.

3 – Proposition

Comme expliqué dans la section 2.4, notre algorithme nommé *Database Matching* repose sur la combinaison de « GUMM » et « MT4MDE », et s'effectue en trois étapes.

3.1 – Conversion de schéma entité/association en graphe

Cette étape consiste à convertir un schéma entité/association (du modèle développé dans le projet des M2 TI) en un graphe orienté et étiqueté conforme à l'approche « GUMM » ; les entités et associations sont converties en nœuds étiquetés portant le nom de l'élément représenté, les attributs quant à eux sont convertis en nœuds portant chacun le nom de l'attribut précédé du nom de l'entité à laquelle cet attribut appartient. Les relations entité-attribut, entité-association et association-attribut sont représentées par des arcs unidirectionnels partant du nœud de l'élément source à celui de l'élément destination.

3.2 – « Similarity Flooding » adapté

L'algorithme de « Similarity Flooding » a été adapté aux schémas entité/association pour pouvoir ainsi utiliser les différentes mesures de similarité servant à améliorer l'exactitude de l'alignement. Cette étape s'effectue désormais en quatre sous-étapes, où les deux premières étapes consistant à créer un graphe de compatibilité et un graphe de propagation restent inchangées, la troisième étape a désormais pour but de calculer la similarité entre les différents nœuds de compatibilité, en utilisant notre propre approche basée sur différentes techniques d'alignement. Toutefois, lors du calcul de la similarité sur un nœud de compatibilité, un processus de filtrage par ressemblance basé sur la « Tokenization » (cf. section 3.2.1) est appliqué aux étiquettes des nœuds à aligner (rappelons que chaque nœud de compatibilité comporte un nœud du graphe source et un nœud du graphe destination, cf. Figure 2). La quatrième et dernière étape de notre algorithme consiste tout simplement à filtrer les correspondances trouvées par rapport au seuil.

3.2.1 – Processus de filtrage par ressemblance

Afin de définir le rôle de ce processus, définissions d'abord le principe de la « Tokenization ». La « tokenization » a pour but de découper une chaîne de caractères en plusieurs sous-chaînes à l'aide d'un ou plusieurs délimiteurs, ces sous-chaînes serviront par la suite d'entrée pour notre processus ; dans le cas des schémas entité/association, il est très fréquent de trouver des délimiteurs comme l'espace, le tiret-bas ou encore l'écriture en « CamelCase ». Le processus de **filtrage par ressemblance**, se déroule en plusieurs étapes :

- La première étape a simplement pour but de récupérer deux listes de mots fournies respectivement par la « tokenization » du premier et du deuxième nœud.
- La deuxième étape sert à éliminer les doublons de chacune des listes de mots récupérées, et à mettre tous les mots en minuscules afin de pouvoir les comparer les uns aux autres.
- La dernière étape consiste à supprimer les mots figurant dans les deux listes pour n'en garder que ceux différents (cf. Annexe 1).

L'intérêt d'un tel processus, est d'exploiter au mieux la sémantique des chaînes de caractères. Pour la suite de l'analyse des nœuds de compatibilités, seuls les mots restants après filtrage par ressemblance sont pris en considération par les techniques d'alignement.

3.2.2 – Techniques d'alignement

Les techniques d'alignement permettent d'attribuer une valeur de similarité pour chaque nœud de compatibilité, cette valeur comprise entre 0 et 1, permettra par la suite de définir si la correspondance est valide ou pas. L'algorithme fournit trois principales techniques d'alignement :

- « *Basique* » : reposant sur la synonymie uniquement, où chaque paire d'éléments à aligner se voit affectée une valeur de similarité égale à 1 si les deux éléments sont synonymes, 0 sinon. Dans le cas d'un alignement d'attributs, ces derniers ne sont étudiés que si les entités auxquelles ils appartiennent le sont également, dans le cas contraire, la valeur 0 est automatiquement attribuée. Cette technique ne prend cependant pas en considération toutes les relations possibles à exploiter lors d'un alignement, malgré la fiabilité qu'elle apporte à l'algorithme. D'où le besoin d'introduire une nouvelle technique plus riche en termes de contraintes.
- « *Améliorée* » : reposant sur la technique « Basique », à laquelle s'ajoutent les contraintes d'intégrité. Cette technique diffère selon le type des éléments à aligner ; dans le cas d'un alignement d'entités, les attributs de ces dernières ainsi que les entités qui leurs sont associées sont comparées (en utilisant la technique « Basique »), afin d'en déduire la valeur de similarité de ces deux entités. Dans le cas d'un alignement d'attributs, la technique « Basique » est effectuée. L'alignement des associations s'effectue en dernier, de la manière suivante : Si les entités reliées par la première association ont été alignées avec celles reliées par la deuxième, la technique Basique est appliquée sur ce nœud de compatibilité afin de lui attribuer une valeur de similarité, sinon ce nœud de compatibilité reçoit une valeur de similarité égale à 0. Hormis les cas précédemment cités, les éléments sont considérés non-alignables. Cette technique optimise les résultats de notre algorithme, cependant, toutes les techniques jusqu'à présent reposent principalement sur la synonymie. Nous pouvons donc améliorer notre algorithme en y introduisant de nouvelles mesures de similarité capables de renforcer la détection de similarité entre les différents éléments comparés.
- « *Complète* » : reposant sur la technique « Améliorée » à laquelle s'ajoutent des mesures terminologiques. Des tests ont été effectués sur plusieurs mesures terminologiques existantes, et nous ont menés à développer une nouvelle mesure basée sur le préfixe (section 3.3), en complément à celle basée sur la distance de Levenshtein.

3.3 – Mesure de similarité par préfixe

Le problème posé dans le cas d'une mesure de similarité par préfixe, est de décider de la valeur à retourner lorsqu'un mot est préfixe de l'autre. Deux réponses à ce problème ont été apportées, la première est de retourner 1 lorsqu'un mot est préfixe de l'autre, 0 sinon, et la deuxième réponse est de retourner une moyenne calculée par la division du nombre de caractères du mot le plus court sur celui du mot le plus long, mais quelle réponse est la plus adaptée à notre besoin ? L'exemple suivant représente le cas où la deuxième réponse n'est pas adaptée à l'alignement des schémas entité/association :

- Soit les deux mots « *dateEnregistrement* » et « *dateEnr* », la première réponse retournera une valeur égale à 1 signifiant que l'un des deux mots est préfixe de l'autre, ce qui est avantageux dans notre cas, tandis que la deuxième réponse retournera 0,38. Cette faible valeur sera omise par notre algorithme à moins que le seuil ne soit très bas (inférieur ou égal à 0,38), et plus la différence entre le nombre de caractères des deux chaînes est plus grande, plus cette valeur retournée est petite, ce qui désavantage cette réponse.

3.4 – Création du schéma intégré

Cette étape prend en entrée l'alignement fourni par l'algorithme de « Similarity Flooding » adapté, au début, un schéma entité/association vide est créé, pour chaque nœud de compatibilité ayant une valeur de similarité supérieure ou égale au seuil, un élément du même type est ajouté au schéma entité/association :

- Si le nœud de compatibilité est composé de deux nœuds représentant des entités, une entité portant le nom du nœud source est créée.
- Sinon, si le nœud de compatibilité est composé de deux nœuds représentant des attributs, un attribut portant le nom du nœud source est créé, et rattaché à son entité -ou association- résultat.
- Sinon, le nœud de compatibilité est donc composé de deux nœuds représentant des associations, une association portant le nom du nœud source est créée, et rattachée aux entités reliées par ces associations.

4 – Validation et expérimentation

4.1 – Description du protocole

Un prototype portant le nom de *Database Matching* a été développé en Java, ce prototype permet de générer un schéma intégré à partir de deux instances de Schéma entité/association. *Database Matching* propose deux interfaces de contrôle, soit un terminal, et dans ce cas les paramètres sont fixés au préalable, soit à l'aide d'une interface graphique, et dans ce cas l'utilisateur peut paramétrer l'outil. (Voir annexe 2). Pour la suite, nous considérons que l'utilisateur lance l'interface graphique du prototype.

L'outil génère une arborescence pour chaque schéma d'entrée, dont les entités et associations sont représentées par des nœuds ayant comme fils leurs attributs. Le seuil et la technique à employer sont des paramètres et peuvent être modifiés selon le besoin. Après lancement de l'algorithme, les résultats sont affichés au fur et à mesure de leur découverte, un résultat est représenté sous forme de ligne dans une table d'alignements, contenant l'élément source, l'élément destination, la valeur de similarité entre ces deux éléments, et une colonne indiquant si la correspondance est considérée possible ou pas. Un compteur d'alignement en bas de l'interface graphique indique l'avancement de l'algorithme.

A la fin de l'exécution de l'algorithme d'alignement, il est possible de générer un schéma intégré (section 3.4) en appuyant sur le bouton correspondant, en bas à droite de l'interface. Il est aussi possible d'évaluer les performances de l'algorithme, et plus précisément de chaque technique utilisée en appuyant sur « Evaluer les performances », un histogramme représentant la précision[2], le rappel[3] et le score[4] est généré, dans ce cas nous distinguons deux cas possibles :

- Si l'alignement correct a été codé dans le programme, l'outil générera un histogramme en fonction de cet alignement, et de celui généré par la technique utilisée.
- Sinon, l'utilisateur peut cocher/décocher les correspondances trouvées afin de modifier l'alignement, dans ce cas-là, les modifications apportées par l'utilisateur seront prises en compte lors de la génération de l'histogramme.

4.1.1 – Synonymie

Afin d'implémenter la mesure de similarité basée sur la synonymie en anglais, le site internet « WordReference » nous a attribué une clé d'accès à une API qui nous permet d'interroger son dictionnaire de synonymes. Pour la mesure de similarité basée sur la synonymie en français, un dictionnaire de synonymes en ligne développé par le « Centre de Recherche Inter-langues sur la Signification en Contexte (CRISCO) » a été utilisé.

Une liste de mots sémantiquement pauvres et propre à chaque langue a été implémentée, et permet de supprimer les mots inutiles lors de la comparaison des éléments, ainsi, des mots comme « dateEnr » et « date_**d'**enregistrement » peuvent désormais être alignés par notre algorithme, car le « **d'** » est filtré.

[2] La précision représente le nombre de correspondances correctement identifiées par l'algorithme, (A), divisé par lui-même + le nombre de correspondances faussement identifiées par l'algorithme, (B). $P = \frac{A}{A+B}$

[3] Le rappel représente le nombre de correspondances correctement trouvées par l'algorithme, (A) divisé par lui-même + le nombre de correspondances non automatiquement identifiées par l'algorithme, (C). $R = \frac{A}{A+C}$

[4] Le score est calculé de la manière suivante : $S = 2 \times \frac{P \times R}{P+R}$

4.1.2 – Filtrage des résultats

L'algorithme de Similarity Flooding, d'où son nom, génère toutes les correspondances possibles entre le graphe source et le graphe destination, il est souvent le cas d'avoir un nombre de résultats supérieur à 100. C'est pourquoi un système permettant de filtrer les résultats selon plusieurs critères a été implanté, afin de minimiser le temps de recherche. Le prototype permet ainsi de :

- Filtrer par « oui » ou par « non » : Lorsque l'utilisateur souhaite se limiter aux résultats ayant été alignés (oui) ou non alignés (non), il peut simplement appuyer sur l'un des deux boutons « Filtrer par oui » et « Filtrer par non ».
- Filtrer par éléments : À partir des arborescences des deux schémas entité/associations affichées en haut du prototype, l'utilisateur peut simplement sélectionner un nœud du graphe source et/ou un nœud graphe destination, le prototype filtrera alors les résultats affichés pour n'en garder que ceux qui ont été choisis.
- Afficher tout : À tout moment, l'utilisateur peut réafficher la liste complète des résultats, en appuyant simplement sur le bouton « Afficher tout », ou en sélectionnant les deux nœuds racines des deux arbres affichés en haut.

4.2 – Performances et résultats

4.2.1 – Tests

Des tests ont été effectués sur le programme en vue de valider notre approche et de la tester sur un jeu de données existant. Les données manipulées lors de ces tests nous ont été fournis par nos encadrants portant sur un projet de l'UE LIF4 (2ème année - Licence informatique, UCBL1). Nous avons également utilisé notre propre jeu de données que nous avons vu lors d'un TP de Mif18 (1ère année - Master Informatique, UCBL1). La plupart des tests ont aboutis, cependant l'algorithme implémenté a des limites, par exemple, lorsque deux mots n'ayant pas de relation de synonymie, et pour lesquels les mesures terminologiques attribuent une faible de similarité, mais signifiant d'un point de vue humain la même chose, ne seront pas alignés automatiquement par l'algorithme, l'exemple trouvé dans un test des données du projet de LIF4 :

- Dans le premier schéma, une entité nommée « Album » contient un attribut « Date_ajout », et dans le deuxième schéma, une entité portant le même nom et contenant un attribut « Date_cree », ces deux attributs n'ont pas été alignés malgré la correspondance trouvée entre leurs entités. Ceci revient au fait que « cree » et « ajout » venant respectivement des verbes « créer » et « ajouter » ne représentent pas des synonymes dans la langue française.

Tous les tests effectués montrent que la technique « Complète » est la plus adaptée aux schémas d'entrée, tandis que les deux techniques « Améliorée » et « Basique » génèrent des scores qui ne sont pas très éloignés.

4.2.2 – Système de cache

Les tests effectués sur le prototype ont montré une optimisation possible au niveau des requêtes envoyées aux dictionnaires implémentés lors d'un alignement, plusieurs requêtes sont redondantes et ralentissent donc le temps d'exécution de l'algorithme comme le montre la figure 6.

Un système de cache permet de garder en mémoire les synonymes trouvés pour chaque mot recherché, et lorsqu'une nouvelle requête se présente, le prototype interroge le système de cache en premier, afin de vérifier si le cache contient une liste de synonymes du mot en question, si tel est le cas, la liste fournie par le cache est prise en considération, et aucune requête supplémentaire n'est générée, sinon, le prototype lance une nouvelle requête, récupère la liste de synonymes et la stocke dans le cache avant de l'utiliser. Chaque langue dispose de son propre système de cache.

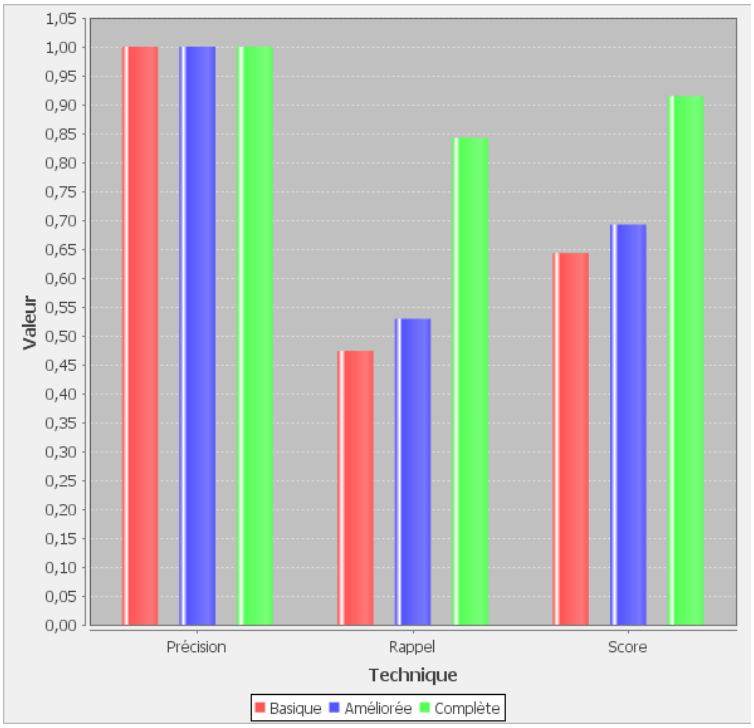


Figure 5 : Comparaison des trois techniques sur un jeu de données de LIF4, dans ce cas, la précision est toujours maximale car aucune mauvaise correspondance n'a été suggérée par les trois techniques d'alignement, ce graphe est généré automatiquement en appuyant sur le bouton « Evaluer les performances » de l'interface graphique.

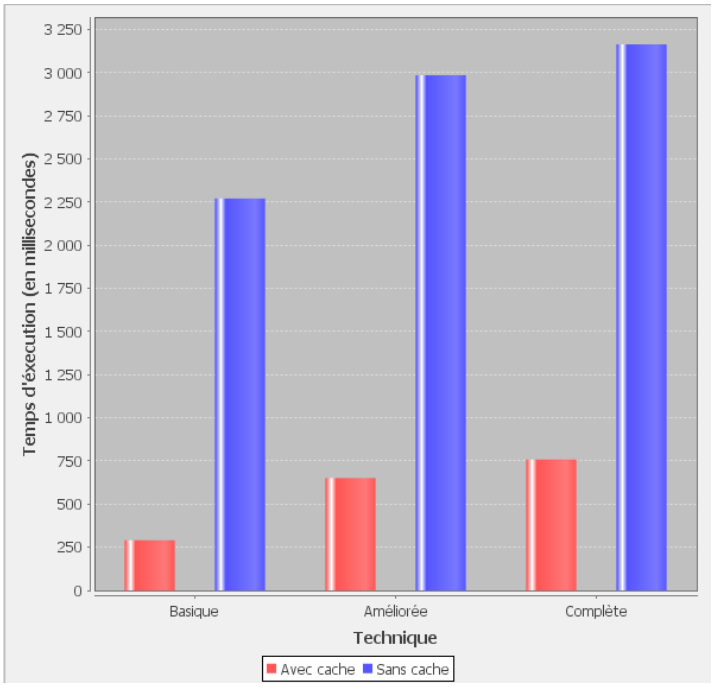


Figure 6 : Comparaison du temps d'exécution des trois techniques sur le même jeu de données, en bleu, le cache n'a pas été utilisé lors de l'exécution, en rouge, le cache a été activé, le gain en termes de temps est trop élevé.

5 – Conclusion

Dans ce papier, nous avons présenté une approche qui permet de générer l'alignement de deux schémas entité/association. Notre contribution peut se résumer en trois principales catégories ; la première étant la mise en œuvre du processus de filtrage par ressemblance qui permet d'exploiter au mieux la sémantique codée dans les chaînes de caractères. La deuxième porte sur les modifications apportées à l'algorithme de Similarity Flooding, ainsi que les différentes mesures de similarité ajoutées et à l'adaptation de ce dernier aux schémas entité/association développés dans un projet de M2 TI. La troisième étant le développement d'un prototype implémentant notre algorithme, afin de pouvoir effectuer des tests de performances dessus. Ces tests ont montré par exemple que la technique Complète est la plus adaptée aux schémas entité/association en termes de précision.

Les principales perspectives que l'on peut citer à l'issue de ce projet de recherche concernent la réutilisabilité de notre travail, afin de réaliser un alignement plus complexe, où un élément appartenant à un schéma, peut être aligné avec un ou plusieurs éléments de l'autre. Exemple : l'attribut « Adresse » d'une entité « X », pourrait être aligné avec l'ensemble d'attributs « Rue », « Code Postal », « Ville » et « Pays » appartenant à une autre entité « Y » (« X » et « Y » sont supposés alignables). Une deuxième perspective consisterait à implémenter une heuristique dans notre algorithme qui lui permettrait par exemple d'apprendre de ses propres erreurs, et de raffiner ses résultats grâce au feedback utilisateur. Une autre perspective orientée pédagogie, consisterait à concevoir un outil d'apprentissage des différentes mesures de similarité existantes dans le domaine de *la mise en correspondance de schémas conceptuels*.

Remerciements

Nous tenons tout d'abord à remercier nos encadrants, Mr Fabien DUCHATEAU et Mr Nicolas LUMINEAU pour leur énorme contribution à ce projet de recherche. Nous tenons également à remercier la responsable de la formation et de l'UE Mif20, Mme Stéphanie JEAN-DAUBIAS, ainsi que les étudiants de Master 2 TI avec qui nous avons collaboré lors de la réalisation de ce projet de recherche. Sans oublier de remercier Mr J.-R. FALLERI de nous avoir fourni le code source du prototype GUMM.

Références

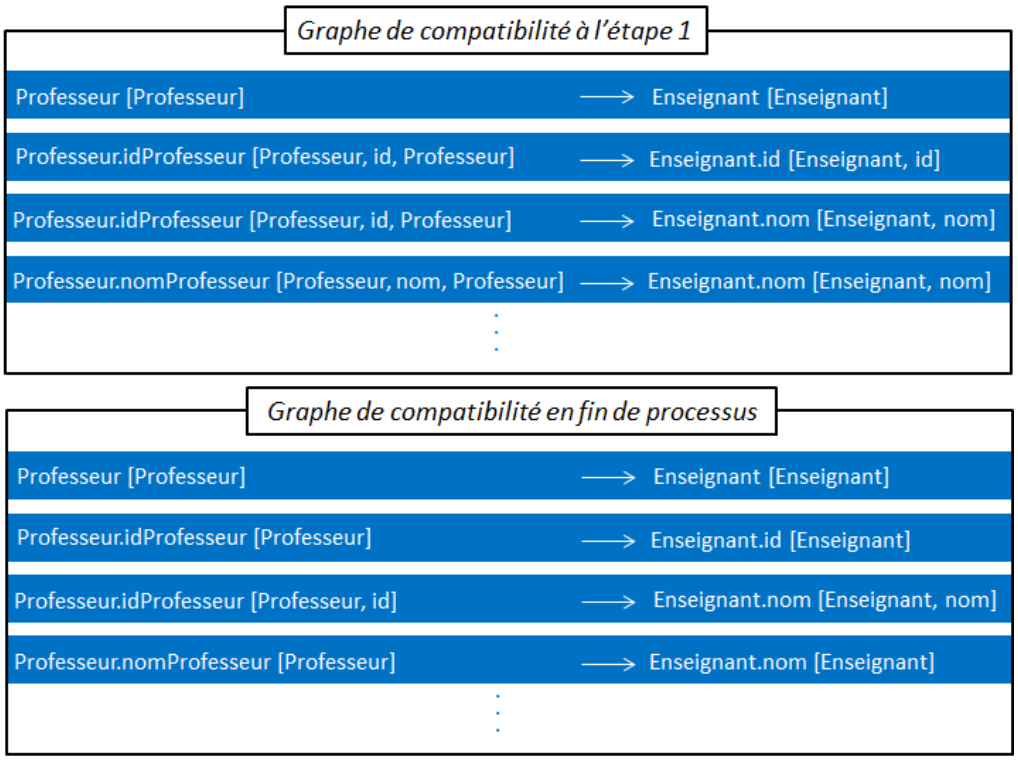
- [JD13]: TER « Application de rétro-ingénierie pour la construction de schémas Entité/Association à partir d'un modèle relationnel ». **J. DECOURSELLE, 2013**, UBCL1, Lyon - France.
- [FHLC08]: « Metamodel Matching for Automatic Model Transformation Generation ». **J.-R. FALLERI, M. HUCHARD, M. LAFOURCADE ET C. NEBUT, 2008**, MODELS'08: 11th International Conference on Model Driven Engineering Languages and Systems, Toulouse - France.
- [LHSB06]: « Metamodel Matching: Experiments and Comparison ». **D. LOPES, S. HAMMOUDI, J. DE SOUZA ET A. BONTEMPO, 2006**, Proceedings of the International Conference on Software Engineering Advances (ICSEA'06), Tahiti – Polynésie Française.
- [MGR02]: « Similarity Flooding, a Versatile Graph Matching Algorithm and Its Application to Schema Matching ». **S. MELNIK, H. GARCIA-MOLINA ET E. RAHM, 2002**, ICDE '02 Proceedings of the 18th International Conference on Data Engineering, IEEE Computer Society Washington DC – USA.
- [BFR11]: « Schema Matching and Mapping ». **Z. BELLAHSENE, A. BONIFATI ET E. RAHM, 2011**. ISBN: 978-3-642-16517-7.
- [ES07]: « Ontology Matching ». **J. EUZENAT ET P. SHVAIKO, 2007**. ISBN: 3-540-49611-4.
- [T11]: « Entity Resolution and Information Quality ». **J.-R. TALBURT, 2011**. ISBN: 978-0-12-381972-7.

Webographie

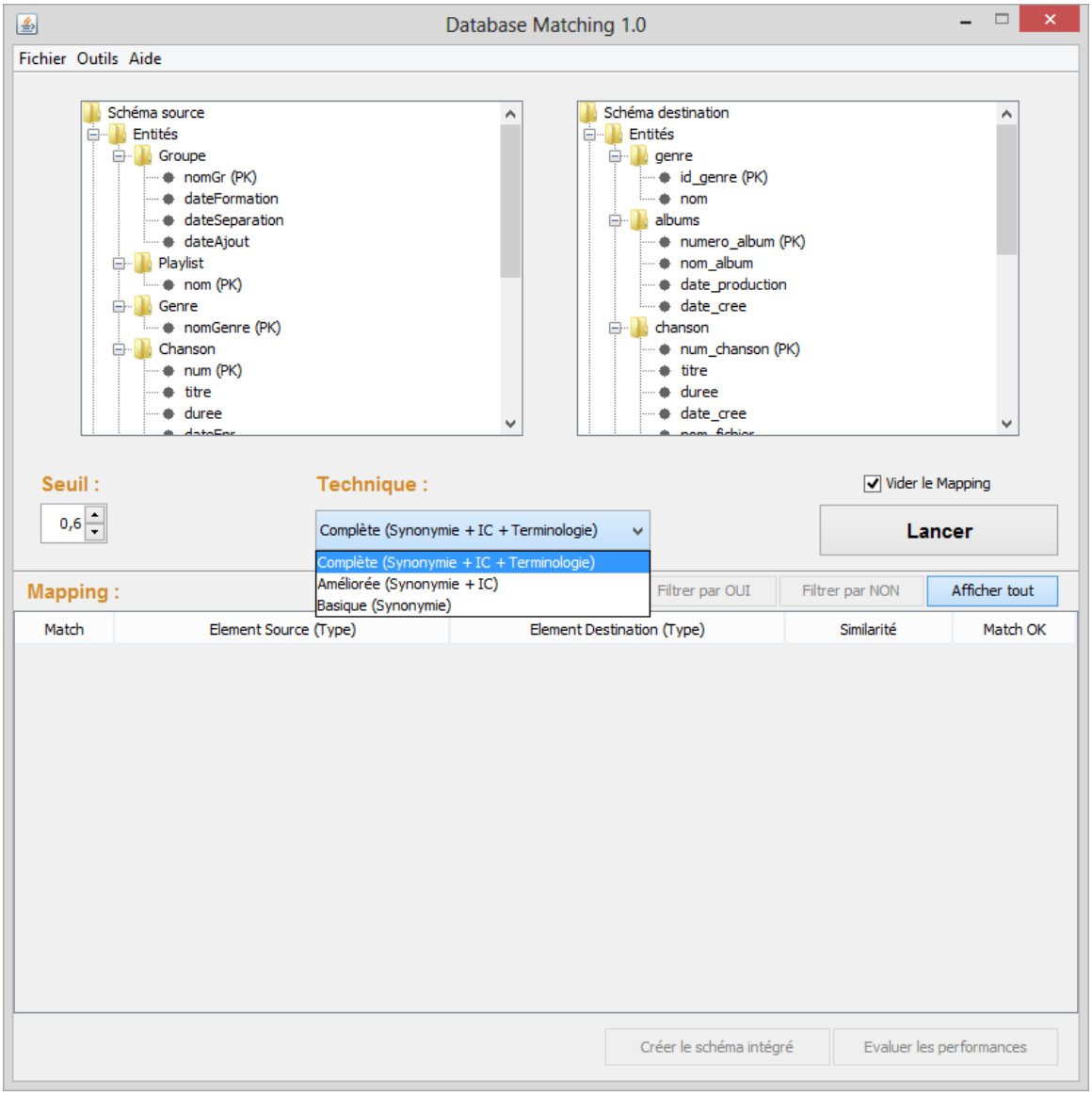
- http://www.wordreference.com/ (API Thésaurus pour l'étude de synonymie en anglais).

- http://www.crisco.unicaen.fr/des/ (Dictionnaire Electronique des Synonymes pour l'étude de synonymie en français).
- http://code.google.com/p/gumm-project/ (GUMM Project)

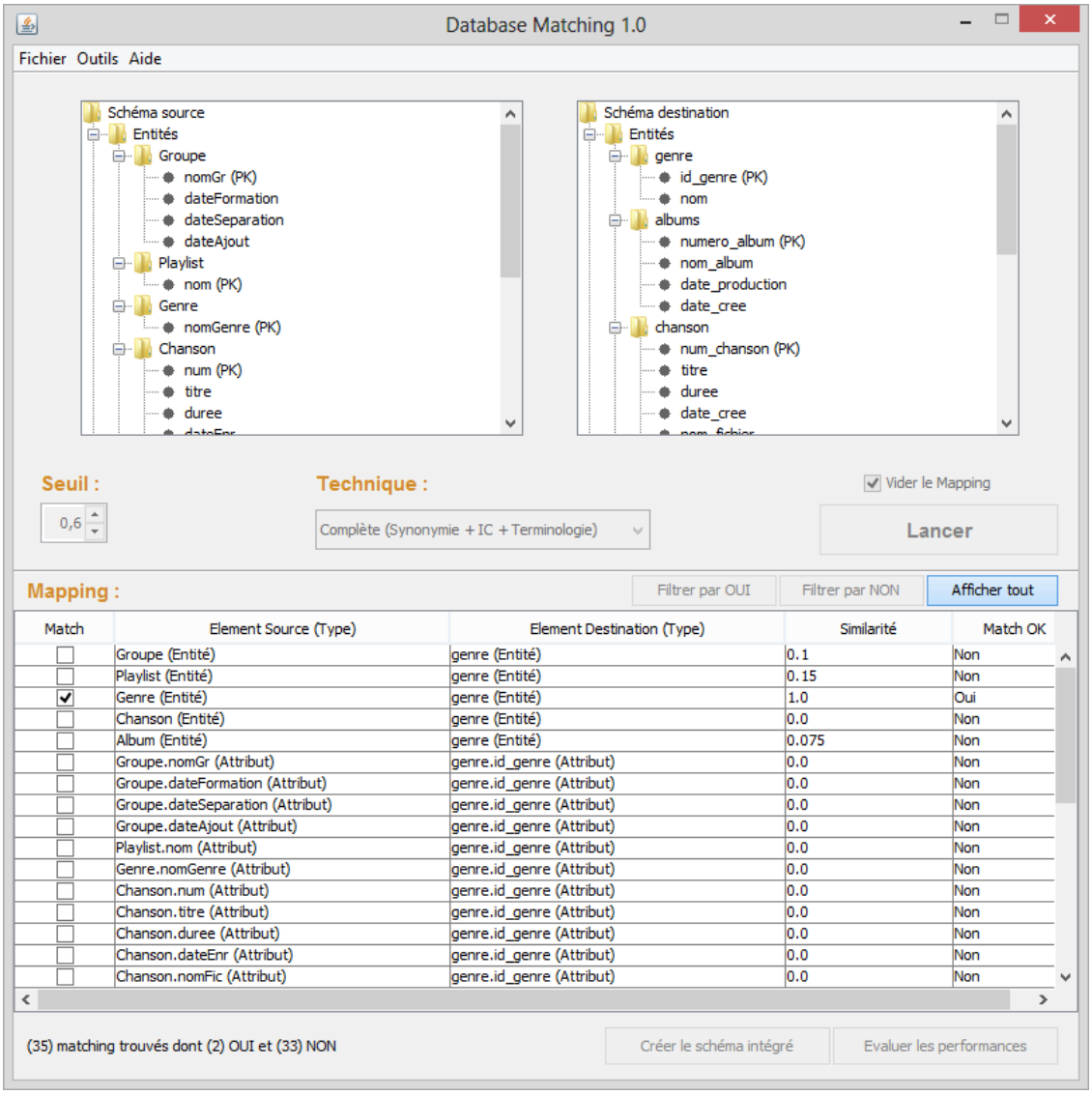
Annexe 1 : Processus de filtrage par ressemblance



Annexe 2 : L'interface graphique de démarrage



Annexe 3 : L'interface graphique lors de l'exécution



Annexe 4 : L'interface graphique en fin d'exécution