MASTER EN SCIENCES INFORMATIQUES

Intégration de données issues du Web

Gillain, Thierry

Award date:
2006

Facultés Universitaires Notre-Dame de la Paix, Namur — Institut d'Informatique.

Année académique 2005 - 2006

Intégration de données issues du Web

Thierry Gillain

Mémoire présenté en vue de l'obtention du grade de Maître en Informatique

Résumé

Aujourd'hui de plus en plus d'information peut être trouvée sur Internet quel que soit le domaine. Cette masse est répartie sur un grand nombre de sites dont le contenu est généralement peu structuré et présente des données tantôt complémentaires tantôt contradictoires. Un site web reprenant l'ensemble des informations d'autres sites, présentées de manière uniforme et dont les incohérences ont été retirées permettrait une consultation plus facile de l'ensemble et permettrait donc de gagner beaucoup de temps.

Dans ce mémoire, nous développerons une méthode complète permettant de produire un site web tel que décrit précédemment. Nous présenterons également le support logiciel qui permettra d'effectuer cette opération. Celui-ci est constitué de logiciels existants qui ont parfois été améliorés et d'autres qui ont été développés entièrement lors de mon stage à l'EPFL.

Mots clé : ontologies, intégration de schémas, intégration de données.

Abstract

Today, very much information can be found on the Internet, about whatever thematic. This information mass is scattered among very much different sites whose content is usually not very structured. The contents of these websites are sometimes complementary, sometimes contradictory. So a unique website containing all the information gathered on the other websites would make life much easier.

In this document, we introduce a methodology and several tools that will allow us to produce such a website from several other ones, all about the same thematic. These tools have two different origins. The first ones are existing tool that have sometimes been improved to perform well in specific parts of the integration process. The second ones have been developped from scratch during my training period at the EPFL.

Keywords : ontologies, schema integration, data cleaning.

Avant-Propos

Je tiens à remercier tout particulièrement :

Monsieur Jean-Luc Hainaut promoteur de ce mémoire pour ses précieux conseils lors de la rédaction de celui-ci.

Monsieur Jean-Roch Meurisse assistant de Monsieur Hainaut pour ses bons conseils tout au long de la rédaction du mémoire et pour avoir relu celui-ci avec attention.

Monsieur Stefano Spaccapietra, directeur du Laboratoire de Bases de données à l'EPFL, de m'avoir accueilli pendant 4 mois dans son laboratoire et m'avoir ainsi permis de mener à bien mon stage.

Mademoiselle Anastasiya Sotnykova, ma maître de stage à l'EPFL, pour m'avoir très bien conseillé et encadré pendant mes 4 mois de stage à l'EPFL.

L'ensemble du personnel du LBD de l'EPFL pour son accueil et sa sympathie pendant mon stage.

L'ensemble de ma famille de m'avoir toujours supporté et encadré pendant toutes mes études ainsi que pour avoir relu attentivement le présent mémoire.

Table des matières

Table des figures

Liste des tableaux

Glossaire

Cible (d'une propriété) Une classe appartient à la cible d'une propriété si cette propriété admet comme valeur un individu de cette classe.

Correspondance Deux éléments appartenant à deux ontologies/schémas différent(e)s sont reliés par une relation de correspondance s'ils représentent des concepts apparentés ou identiques du monde réel.

Domaine (d'une propriété) Une classe appartient au domaine d'une propriété si une ou des valeurs de cette propriété sont susceptibles d'être associées à un individu de cette classe.

Individu Correspondant dans les ontologies des instances de classes dans les langages orientés objets.

InverseFunctional (propriété) Si une propriété est InverseFunctional, cela signifie que sa propriété inverse est fonctionnelle (sa cardinalité maximale est de 1 pour toutes les classes auxquelles elle est associée).

Intégration des instances/individus Cette opération consiste à découvrir parmi un ensemble d'instances/individus, celles/ceux qui représentent le même objet du monde réel.

Mapping cf. correspondance.

Monde Fermé (hypothèse du) Cette hypothèse considère que toute connaissance non donnée explicitement est fausse. Elle est notamment en vigueur dans les bases de données.

Monde Ouvert (hypothèse du) Cette hypothèse considère que toute connaissance non donnée explicitement est inconnue. Elle est notamment en vigueur dans les ontologies.

Namespace un espace de nommage (namespace en anglais) est une notion permettant de désambiguïser des termes qui pourraient sinon être homonymes. Un espace de nommage est matérialisé par un préfixe identifiant de manière unique la signification d'un terme. Au sein d'un même espace de nommage, il n'y a pas d'homonymes.

Ontologie Selon T. Gruber, il s'agit de la spécification formelle d'une conceptualisation. De manière plus simple, nous définissons une ontologie comme la représentation, commune à un ensemble d'experts, des concepts d'un domaine et de leurs relations définis dans un langage formel dont la sémantique est comprise sans équivoque. Les concepts sont généralement organisés sous la forme d'une taxonomie (une hiérarchie d'héritage).

OWL Web Ontology Language est un dialecte XML basé sur une syntaxe RDF. Il fournit les moyens pour définir des ontologies Web structurées. Le langage OWL est basé sur la recherche effectuée dans le domaine de la logique de description. OWL peut être vu en quelque sorte comme un format de fichier pour certaines logiques de descriptions. OWL permet de décrire des ontologies, c'est-à-dire qu'il permet de définir des terminologies pour décrire des domaines concrets. Une terminologie est composée de concepts et de propriétés (aussi appelés rôles en logiques des descriptions). Un domaine se compose d'instances de concepts

RDF Resource Description Framework est un modèle de graphes pour décrire les (méta-)données et permettre un certain traitement automatique des métadonnées. Une des syntaxes (sérialisation) de ce langage est RDF/XML. Il s'agit d'un dialecte XML développé par le consortium W3C.Un document structuré en RDF est un ensemble de triplets. Un triplet RDF est une association : sujet, objet, prédicat ce qui signifie ($\forall$objet,$\exists$sujet *tq* predicat(sujet,objet))

Schéma/ontologie global(e) Schéma ou ontologie résultant de l'intégration de plusieurs autres schémas/ontologies.

Taxonomie Les taxonomies sont également appelées hiérarchies d'héritage. Les taxonomies sont représentées sous la forme d'arbres dont les noeuds sont les classes et les arcs sont les relations de sous-classes existant entre les classes.

URI Un URI, de l'anglais Uniform Resource Identifier, soit littéralement identifiant uniforme de ressource, est une courte chaîne de caractères identifiant une ressource physique ou abstraite, et dont la syntaxe respecte une norme d'Internet mise en place pour le World Wide Web.

Chapitre 1

Introduction

1.1 Problématique

Aujourd'hui de plus en plus d'information peut être trouvée sur Internet quel que soit le domaine. Cette masse est répartie sur un grand nombre de sites. Le contenu de ces derniers est généralement peu structuré et présente des données tantôt complémentaires tantôt contradictoires. De plus si un utilisateur souhaite obtenir de l'information sur un thème particulier (cours de bourse, choix d'un livre à acheter,...), dans l'état actuel des choses, les seuls moyens dont il dispose pour l'aider dans sa tâche sont les moteurs de recherche ainsi que les marques pages mémorisés dans son navigateur. Les différents sites qu'il sera amené à consulter possèderont en général une présentation relativement différente, ceci ne facilitant pas la comparaison de l'ensemble des sites et ne permettant donc pas d'avoir une vue globale de l'information qui y est présentée.

Cette tâche pourrait être grandement facilitée s'il existait un site regroupant les données d'autres sites dont les incohérences auraient été supprimées et fournissant une présentation homogène de l'ensemble.

La production d'un tel site n'est cependant pas chose aisée. En effet, chaque site offre des informations qui lui sont spécifiques et d'autres qui sont présentes sur différents sites parfois sous des vocables identiques, parfois sous des vocables différents. Il arrive même qu'un concept possède plusieurs valeurs sur l'un des sites, alors qu'il n'en possédera qu'une sur un autre. Il sera donc important dans la construction de notre site unifié de repérer les concepts communs afin de les regrouper sous un même vocable sur ce site. Il nous faudra aussi régler les conflits de structure comme le nombre de valeurs autorisées sur le site unifié pour un concept particulier.

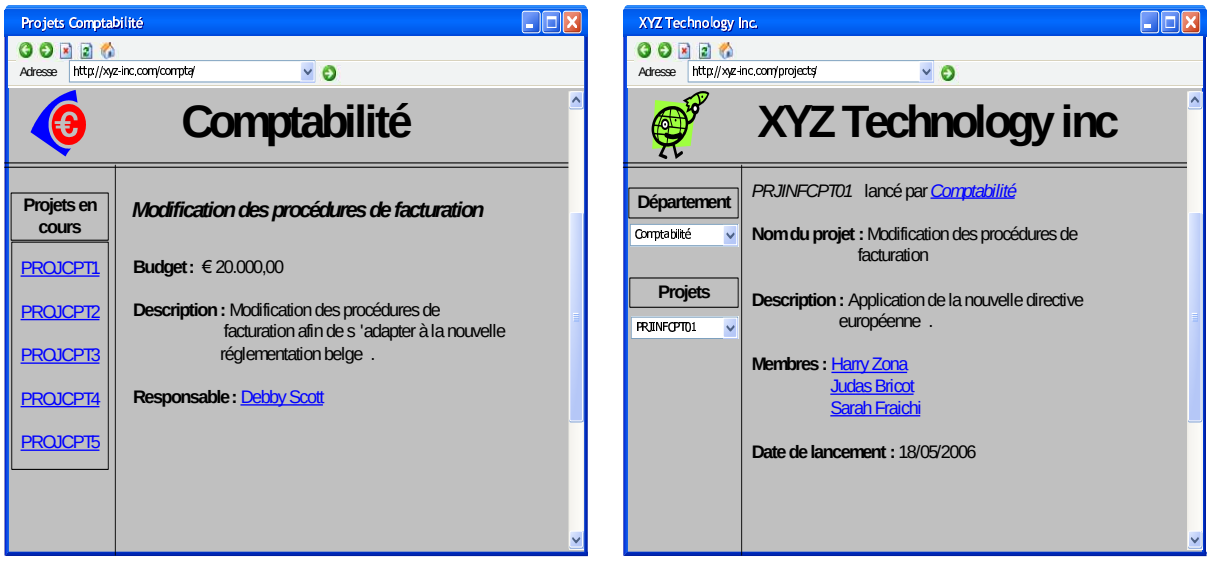
Notons également que des sites différents peuvent contenir des informations pour un même objet du monde réel (ex : un même projet comme dans la section 1.2). Lorsqu'une telle situation se présente, différents cas sont à distinguer, les informations fournies par les différents sites peuvent être complémentaires, être équivalentes ou être en conflit. Le conflit entre deux valeurs peut avoir deux origines, soit l'une des valeurs est erronée soit les deux valeurs sont valides pour cet objet (ex : un prix). Il faut dès lors être à même de repérer ces objets identiques, d'en corriger ou d'en accepter les conflits afin de pouvoir combiner leurs informations sur le site web unifié.

1.2 Brève illustration de la méthode

Afin de permettre au lecteur d'avoir un bon aperçu du processus d'intégration, avant de se plonger de manière approfondie dans la lecture de ce mémoire, celui-ci sera brièvement illustré ci-dessous sur un exemple très simple.

1.2.1 Mise en situation

Supposons deux sites présentés à la figure 1.1. Tous deux sont des sites de gestion de projets au sein des départements de l'entreprise *XYZ Technology inc.*. Le premier site (a) a été réalisé par Monsieur Dupont le directeur du département comptabilité afin de mieux gérer les projets en cours dans son département. Ce site fournit les budgets alloués à chacun des projets, et le nom de la personne responsable de ceux-ci. Plus tard un deuxième site (b) a été créé suite au désir de l'entreprise d'informatiser et de centraliser la gestion de projets des différents départements. Le problème est que ce nouveau site a été fait sans tenir compte des desiderata individuels de chacun des directeurs de départements. Monsieur Dupont est convaincu de l'intérêt de la centralisation et veut mettre à la disposition de tous les informations concernant les projets en cours dans son département. Cependant, il lui semblerait bon de posséder non seulement les informations données par le site général mais également des données plus spécifiques présentes sur son propre site comme le budget ou le responsable. De plus il remarque que les informations fournies par les deux sites ne sont pas toujours identiques et voudrait pouvoir comparer facilement celles-ci.



(a) (b)

FIG. 1.1 – Les sites de gestion de projets (a) du département comptabilité, (b) de l'entreprise.

1.2.2 Extraction de la structure et des données

Les concepts importants des deux sites sont définis et structurés cf. figure 1.2) à l'aide du plugin du navigateur web Mozilla, Retrozilla développé à l'Université de Namur [Estievenart et al., 2006]. Les données sont extraites dans un format correspondant à la structure définie par le plugin.

1.2.3 Recherche des correspondances et intégration

Les structures sont ensuite comparées entre elles afin de repérer les concepts communs et de pouvoir les regrouper sous un même vocable dans une structure unifiée. Les correspondances entre

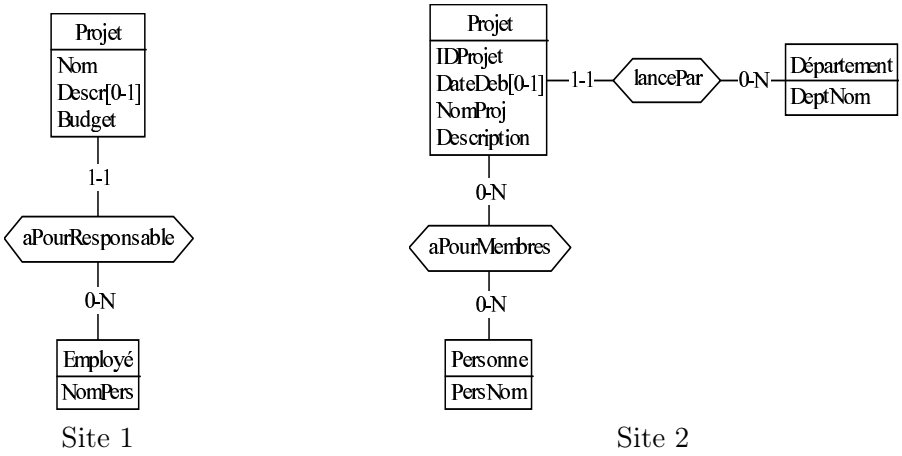


FIG. 1.2 – Les structures sous-jacentes des concepts des deux sites de gestion des projets.

les deux structures sont ensuite représentées au moyen de règles (lignes bleues sur la figure 1.3). La structure unifiée est présentée à la figure 1.4. Dans la méthode développée ici, la comparaison et l'intégration sont effectuées de manière semi-automatique par un expert du domaine. Celui-ci pouvant compter sur une aide logicielle pour lui suggérer les actions à entreprendre.

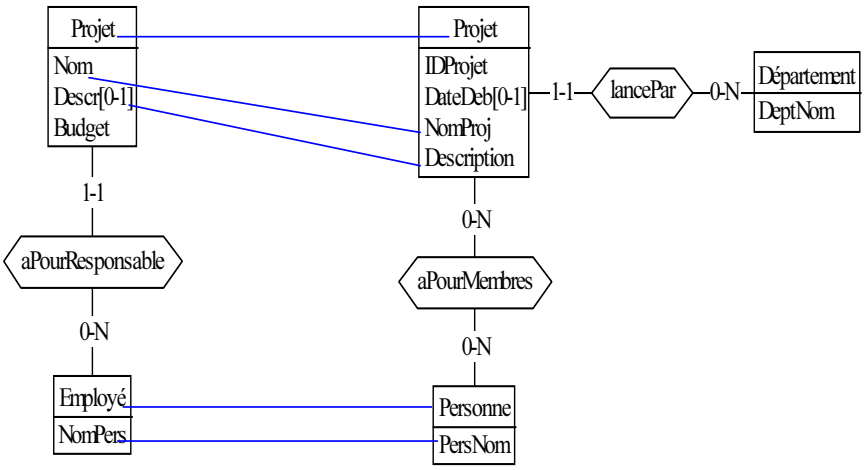


FIG. 1.3 – Les correspondances découvertes entre les structures des deux sites web.

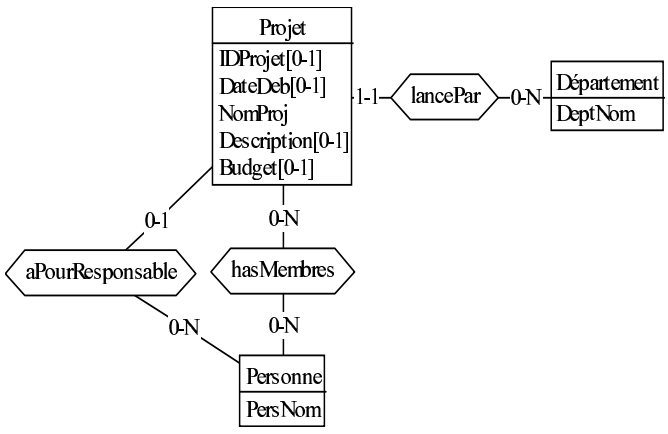


FIG. 1.4 – La structure unifiée des deux sites web d'origine.

1.2.4 Exportation des données

Nous disposons à présent d'une structure unifiée des deux sites. Pour que celle-ci puisse servir au but d'intégration que nous nous sommes fixé, il faudrait qu'elle contienne également les données présentes dans les sites d'origine. Pour cela, il nous faut disposer des correspondances entre les concepts des structures des sites et la structure unifiée afin de pouvoir traduire les données dans un format conforme à la nouvelle structure.

1.2.5 Intégration des données

Si l'on observe les pages des deux sites d'origine présentées à la figure 1.1(a,b), on remarque qu'elles représentent en réalité le même projet vu sous deux angles différents. Ces différentes vues peuvent conduire pour une même propriété du projet à des valeurs identiques (ex : le nom des projets), complémentaires (le responsable du projet fourni par un seul des sites) ou conflictuelles (la description du projet). Un tel conflit peut dans certains cas provenir d'une erreur, ou dans d'autres cas fournir une autre valeur tout aussi correcte pour la propriété (ex : la description du projet qui est différente au niveau de l'entreprise et du département). Il nous faut donc lors de cette étape repérer toutes les paires de projets identiques et résoudre ou accepter les conflits entre les différentes valeurs pour une même propriété.

1.2.6 Affichage de la structure unifiée

La dernière étape consiste à redéployer les données de la structure globale dans un format lisible par un utilisateur. Etant donné les sources web, l'utilisation d'une interface web semblait la plus pertinente pour l'affichage des données de la structure unifiée. Comme cela a été dit dans la section précédente, l'affichage de l'ontologie globale doit présenter toutes les informations que l'on possède sur un projet particulier. Un exemple de page web articulant les données du projet de la figure 1.1 est présenté à la figure 1.5.

FIG. 1.5 – Affichage des données unifiées des projets d'origine.

1.3 Contenu du mémoire

Dans ce mémoire, nous développerons une méthode complète permettant de produire un site web unifié à partir de plusieurs autres sites. Nous présenterons également le support logiciel qui permettra d'effectuer cette opération. Celui-ci est constitué de logiciels existants et d'autres développés pour supporter des parties spécifiques du processus d'intégration.

Les contributions de ce mémoire sont les suivantes :
- Elaboration d'une méthode permettant d'intégrer les structures et les données provenant de différents sites web.
- Définition d'une méthode permettant de découvrir les instances représentant le même objet du monde réel au sein de différents sites afin de combiner leurs informations lors de l'affichage.
- Implémentation des deux méthodes précédentes avec un but d'automatisation maximale.
- Développement d'une application permettant d'afficher de manière homogène les données de l'ontologie issue de l'intégration de différents sites.

La suite du présent mémoire sera organisée comme suit :

Dans une première partie, nous présenterons les notions essentielles à une bonne compréhension de la méthode développée par la suite. Pour cela, nous commencerons dans le chapitre 2 par expliquer ce qu'est une ontologie et en quoi ce type de représentation de connaissances diffère des bases de données. Nous présenterons également certains outils utiles pour manipuler des ontologies ainsi que des langages de représentation d'ontologies. Dans le chapitre 3, nous nous attarderons davantage sur l'intégration d'ontologies, sur les différentes étapes nécessaires à ce processus. Nous développerons aussi les différentes manières d'aborder l'intégration de N schémas/ontologies et sur les différentes architectures existant afin de mettre en relations les données des schémas/ontologies d'origine avec le schéma/l'ontologie global(e). Nous approfondirons ensuite davantage l'étape de recherche des correspondances entre différents schémas/ontologies lors de leur intégration. Enfin, nous présenterons un état de l'art des outils et méthodes d'intégration de schémas/ontologies ainsi que des méthodes et outils d'intégration des instances (recherche des instances représentant un même objet du monde réel).

Dans une deuxième partie, nous commencerons dans le chapitre 4 par décrire de manière générale la méthodologie d'intégration que nous avons adoptée dans ce mémoire, pour ensuite examiner plus en détail chacune des étapes du processus. Dans le chapitre 5, nous présenterons les outils de support utilisés lors des différentes étapes de la méthode. Enfin, dans le chapitre 6, nous illustrerons notre méthode en l'appliquant sur un cas réel.

Première partie

Concepts généraux

Chapitre 2

Les ontologies

2.1 Qu'est-ce qu'une ontologie ?

Le terme ontologie est à l'origine issu de la philosophie où il désigne *l'étude de l'être en tant que être*, il s'agit donc de l'étude des propriétés générales de ce qui existe. Bien que tiré de racines grecques, le terme n'est apparu officiellement qu'au XVII siècle. Ce terme a ensuite été repris par les chercheurs du monde de l'Intelligence Artificielle qui considèrent que *ce qui existe* équivaut à ce qui peut être représenté. T. Gruber a alors proposé la définition suivante :
"*An ontology is a formal explicit specification of a shared conceptualization.*"

De manière plus simple, nous définissons une ontologie comme la représentation, commune à un ensemble d'experts, des concepts d'un domaine et de leurs relations définis dans un langage formel dont la sémantique est comprise sans équivoque. Les concepts sont généralement organisés sous la forme d'une taxonomie (une hiérarchie d'héritage).

Le regain d'intérêt actuel pour les ontologies vient du désir de rendre des connaissances compréhensibles et traitables aisément par une machine. Le principal exemple de cette démarche est le *Web Sémantique*[1]. Cette approche permettrait de faciliter grandement les recherches sur Internet. En effet, le problème des moteurs de recherches actuels est qu'ils nécessitent une intervention humaine afin de trouver la/les page(s) intéressante(s) parmi la longue liste délivrée par le moteur. Il y a en effet très peu de chances pour que cette liste corresponde exactement aux attentes de l'utilisateur. Le problème principal des pages web est qu'elles ne contiennent que peu d'informations concernant la sémantique des informations présentées. Le but du *Web Sémantique* est d'ajouter de la sémantique aux pages web afin de faciliter la découverte automatique des pages répondant aux critères d'une requête. Ceci est généralement effectué par annotation des concepts des sites web à l'aide d'ontologies communément acceptées. Les ontologies peuvent également être utiles pour beaucoup d'autres buts, tels que l'organisation des rubriques d'un site et de leur contenu de manière hiérarchique (ex : Yahoo) ou la vérification de la cohérence des informations par rapport aux spécifications des différents concepts d'une ontologie. L'un des principaux avantages des ontologies est la possibilité de les utiliser avec un moteur d'inférence afin d'en vérifier la cohérence ou de découvrir de nouvelles connaissances à partir des connaissances déjà présentes.

2.2 Composantes d'une ontologie

La question de la définition du contenu d'une ontologie fait encore l'objet de nombreuses discussions. Cependant selon [Uschold and Gruninger, 2004], la plupart des experts s'accordent sur le noyau essentiel d'une ontologie :

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

– Un ensemble de termes importants d'un domaine.
– Une certaine spécification de la sémantique de ces termes au moyen d'une logique.

Selon [Modica, 2002], les constructions disponibles pour créer une ontologie sont en général les suivantes : une illustration de celles-ci est proposée à la figure 2.1.

Des classes. Elles représentent un ensemble d'objets similaires qui partagent des caractéristiques communes (les individus). Elles peuvent également être organisées sous la forme de taxonomies (hiérarchies d'héritage).

Des individus. Les individus d'une classe constituent la population recouverte par cette classe, ce sont les objets qui satisfont aux conditions d'appartenance à cette classe.

Des propriétés Elles décrivent les caractéristiques d'une classe. Elles peuvent être vues comme des fonctions spécialisées. Elles peuvent également être organisées selon une hiérarchie et posséder des attributs particuliers (fonctionnelles, symétriques,...). Une propriété prend en argument un individu d'une classe nommée *domaine* et met cet individu en correspondance avec un individu d'une autre classe nommée *cible*.

Des fonctions Celles-ci sont une généralisation des propriétés et peuvent posséder plusieurs arguments.

Des axiomes Ils sont utilisés pour représenter des relations difficiles à exprimer au moyen de *slots*. Ils expriment, par exemple, des contraintes, des connaissances sur le domaine, des implications, ... et sont en général exprimés au moyen d'opérateurs logiques ($\forall$,$\exists$,$<$,$>$,...).

Une possibilité de composition. La composition est similaire au principe des *types utilisateurs* dans un langage de programmation et permet de regrouper des termes apparentés en une nouvelle classe. Ce regroupement peut s'effectuer par exemple sur la base des valeurs d'une propriété ou de critères sémantiques.

2.3 Ontologies vs. Bases de données

Comme le disent les auteurs de [Uschold and Gruninger, 2004], les bases de données et les ontologies possèdent des caractéristiques communes au point de vue de l'expressivité[2], toutes deux définissent un vocabulaire qui a un sens dans le langage naturel. Toutefois des différences spécifiques existent et sont présentées dans [Shvaiko and Euzenat, 2005] et [Uschold and Gruninger, 2004].

Tout d'abord leurs raisons d'existence sont différentes. Une ontologie est créée notamment pour donner une spécification d'un domaine (par exemple pour une application) ou pour faciliter l'interopérabilité entre plusieurs domaines ou applications. Pour sa part, un schéma de base de données est avant tout créé pour structurer un ensemble de données, afin de pouvoir l'interroger facilement via des requêtes.

Ensuite, les contenus informationnels de leurs "schémas" ne sont en général pas équivalents. Le schéma d'une base de données donne généralement peu d'informations quant à la sémantique de son contenu. Celle-ci est définie au moment de l'analyse conceptuelle mais n'est plus explicite par la suite. Alors que la structure d'une ontologie est avant tout prévue pour exprimer et mettre en avant la sémantique des concepts et relations entre ceux-ci. Ajoutons que les schémas de bases de données sont généralement conçus pour un usage unique, tandis que les ontologies sont évolutives et partagées entre différents usages. Le développement d'un schéma de base de données est en général un processus centralisé alors qu'une ontologie est conçue de manière décentralisée et coopérative.

[2]Le lecteur intéressé trouvera dans l'annexe A une proposition de correspondances entre les ontologies et le modèle entités-associations.

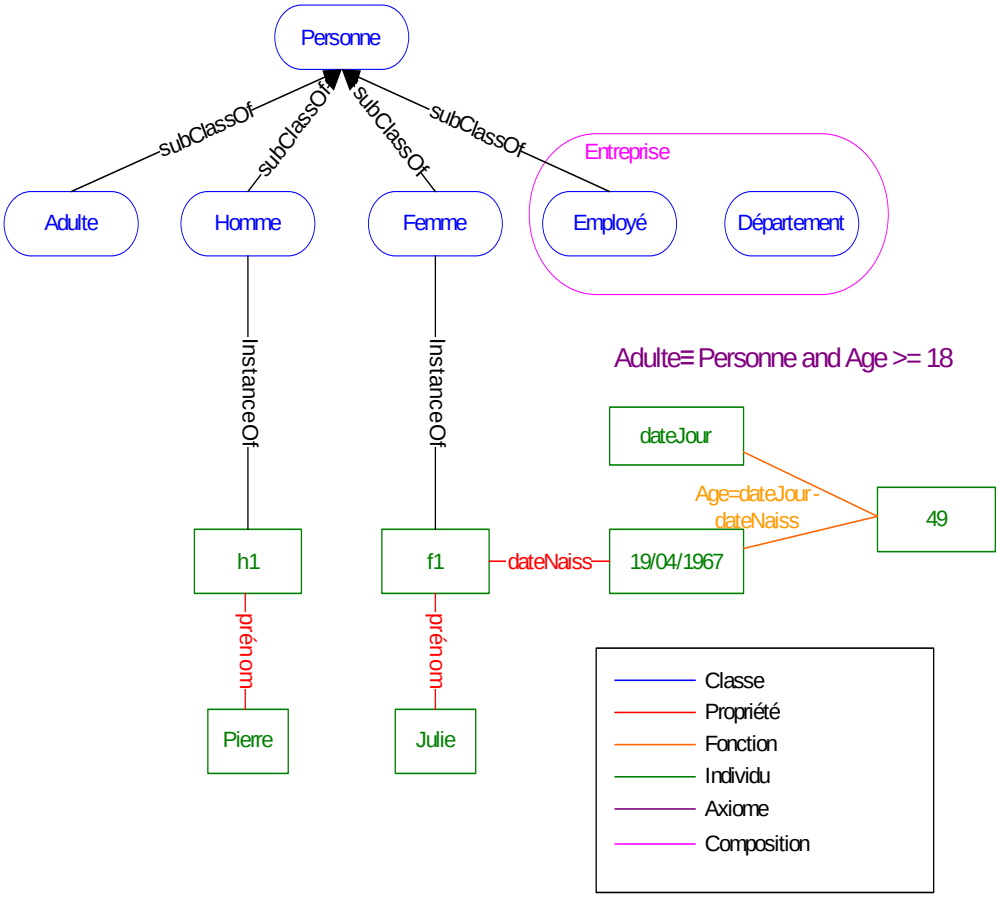


FIG. 2.1 – Illustration des composantes traditionnelles d'une ontologie.

Les rôles des contraintes (axiomes pour une ontologie) sont également différents. Dans une ontologie, les axiomes servent à définir la signification des classes et de leurs relations afin de permettre un traitement automatisé et accessoirement la vérification d'intégrité d'une base de connaissances. Au contraire, pour les bases de données, les contraintes sont définies essentiellement afin de maintenir l'intégrité des données et d'optimiser les requêtes formulées par les utilisateurs. Par exemple, si dans une base de données, les contraintes de cardinalités et de suppression sont essentielles pour assurer la validité des informations présentes dans la base de données et pour éviter toute apparition d'incohérences, les systèmes de gestion d'ontologies sont par contre généralement assez laxistes et ne vérifient pas cette cohérence. De plus, lors de la conception d'une base de données, certaines contraintes trop coûteuses à mettre en oeuvre sont abandonnées lors de sa mise en place sur un système de gestion de bases de données . Ceci ne sera pas le cas dans une ontologie, le but étant de mémoriser le plus d'informations possible sur la sémantique des données pour permettre un traitement par une machine.

En outre, les systèmes de gestion de bases de données sont très spécialisés et optimisés afin de répondre aux requêtes et assurer la cohérence des données. Les systèmes de gestion d'ontologies, s'ils peuvent être utilisés pour répondre à des requêtes ou pour assurer la cohérence, sont tout d'abord conçus pour effectuer des inférences sur les connaissances afin d'en dégager de nouvelles. Ce type d'opération permet par exemple de produire une taxonomie de classes si celle-ci n'était pas définie ou d'affiner une taxonomie existante, ce qui est rarement le cas pour les systèmes de gestion de bases de données .

Enfin, une importante différence entre les ontologies et les bases de données, qui se marque

lors de l'exécution de requêtes, est la conception qu'elles ont du monde. En effet, les bases de données fonctionnent sur l'hypothèse d'un *Monde Fermé* alors que les ontologies se basent sur l'hypothèse d'un *Monde Ouvert*. L'hypothèse d'un *Monde Fermé* considère que toute connaissance non donnée explicitement est fausse. L'hypothèse d'un *Monde Ouvert* quant à elle considère que toute connaissance non donnée explicitement est inconnue. Si l'on considère l'ensemble des auteurs ayant écrit sur un sujet (cf tableau 2.1), la requête "*Durand a-t-il écrit un ouvrage sur OWL ?*" fournira comme réponse *non* dans le cas d'un *Monde Fermé*, alors que la réponse sera *je ne sais pas* si nous travaillons dans un *Monde Ouvert*.

Ouvrage	
Auteur	**Sujet**
Dupont	OWL
Durand	ERA
Durand	SQL
Dupont	ERA

TAB. 2.1 – Table représentant des ouvrages pour lesquels on connaît l'auteur et le sujet.

2.4 Outils et langages de support aux ontologies

Différents langages ont été définis afin de pouvoir représenter des connaissances et des ontologies. Parmi ceux-ci nous pouvons citer les Description Logics, DAML+OIL, RDF, RDF-S, OWL, OBO. Divers outils de manipulation d'ontologie et d'inférence ont également été développés. Dans cette section, nous présenterons les langages et outils qui nous seront utiles dans la méthode d'intégration proprement dite développée au chapitre 4.

2.4.1 Description Logics

Description Logics (DLs) est le nom le plus récent donné à une famille de langages de représentation de la connaissance d'un domaine par la définition de ses concepts et de leurs propriétés. La différence des DLs par rapport à leurs prédécesseurs (réseaux sémantiques, KL-ONE, frame-based)[Baader et al., 2003] est qu'elles offrent une sémantique formelle fondée sur la logique. Une autre caractéristique importante de la logique des descriptions est la possibilité d'inférence de nouvelles connaissances à partir des connaissances déjà présentes dans la base de connaissances.

Les *Description Logics* possèdent un vocabulaire spécifique (cf. section 2.2) qui diffère de celui des ontologies . Le tableau 2.2 présente les correspondances entre ces deux vocabulaires.

DLs	**Ontologies**
Concept atomique	Classe sans conditions d'appartenance particulières (par opposition aux concepts composés).
Concept composé	Classe possédant des conditions d'appartenance.
Rôle	Propriété
Instance	Individu

TAB. 2.2 – Correspondances entre le vocabulaire des DLs et celui des ontologies.

Ce formalisme se base sur une classification des concepts en les organisant sous forme d'une hiérarchie d'héritage ou taxonomie (un concept est un sous-concept d'un autre). Chaque concept contient un ensemble d'instances qui sont conformes aux conditions d'appartenance à un concept (ex : le concept *Mère* pourrait se définir comme l'ensemble des femmes possédant au moins un enfant). Le principe d'héritage s'applique également aux instances. Les instances d'un sous-concept forment un sous-ensemble des instances des super-concepts (ex : les mères font partie de l'ensemble des femmes, mais toutes les femmes ne sont pas mères).

Les bases de connaissance exprimées en DL possèdent deux composants principaux : les TBoxes et les ABoxes. Les TBoxes décrivent les concepts et leurs rôles alors que les ABoxes décrivent les instances.

Chaque langage de DL a un degré d'expressivité spécifique, qui est fonction des constructeurs disponibles dans ce langage et des contraintes imposées sur ces constructeurs.

L'ensemble des constructeurs existant parmi les DLs pour définir des concepts et combiner les conditions d'appartenance aux concepts sont l'union, l'intersection, le complément, les cardinalités d'un rôle, la valeur d'un rôle, le prédicat universel, le prédicat d'existence, le concept universel (contenant tous les individus de toutes les classes → le super-concept de tous les concepts) et le concept vide (ne contenant aucune instance). Notons, que l'ensemble de ces constructeurs n'est pas nécessaire pour garantir l'expressivité maximale. Certaines sont présentes afin de faciliter l'écriture mais pourraient être exprimées par la combinaison d'autres constructeurs (ex : le prédicat universel peut s'exprimer à l'aide du prédicat d'existence et du complémentaire $\forall X.y \equiv \neg\exists X.\neg y$ (où X est un concept et y est une condition d'appartenance à ce concept, $\neg$ exprimant le complémentaire de l'expression suivant le symbole).

L'expressivité de certains sous-langages est parfois également limitée par les contraintes imposées aux constructeurs (on n'utilise qu'une partie des possibilités offertes par les constructions). Le prédicat existentiel peut par exemple n'accepter que le concept universel comme condition.

Le nom d'un langage de DL représente l'ensemble des constructions qu'il contient. Par exemple, le langage de *Description Logic*, $\mathcal{AL}$ (Attributive Language), dont la syntaxe de base est présentée à la figure 2.2, est un langage de description contenant le nombre minimal de constructions pour être utiles dans la pratique

C,D	$\longrightarrow$	A	|	Concept atomique
		$\top$	|	Concept Universel
		$\bot$	|	Concept Vide
		$\neg$A	|	Négation Atomique
		C$\sqcap$D	|	Intersection
		$\forall$R.C	|	Prédicat universel
		$\exists$R.T		Prédicat Existentiel limité

FIG. 2.2 – Les règles de production du langage $\mathcal{AL}$. A représente les concepts atomiques (non décomposables), C et D représentent les concepts composés (construits à l'aide des règles de production), R représente les rôles atomiques.

Par la suite, ce langage a été étendu à l'aide de nouveaux constructeurs chacun représenté par une lettre qui figure dans le nom du langage. Ces constructeurs sont :

– Le prédicat existentiel complet qui est représenté par $\mathcal{E}$. Par exemple : Père ≡ Personne ⊓ SexeMasculin ⊓ ∃estParentDe.Personne.

- Les restrictions de cardinalités sur les propriétés d'un concept ($\geq$n.R et $\leq$n.R où n est un nombre) représentées par $\mathcal{N}$. Par exemple : MèreFamilleNombreuse $\equiv$ Mère $\sqcap \geq$3estParentDe.
- Le complémentaire de n'importe quel concept au lieu des simples concepts atomiques comme dans le langage $\mathcal{AL}$ de base ($\neg$C). Celui-ci est représenté par $\mathcal{C}$. Par exemple : Père $\equiv \neg$ SexeFéminin $\sqcap$ Parent.
- L'union de concepts (C$\sqcup$D) représentée par $\mathcal{U}$. Par exemple : Parent $\equiv$ Père $\sqcup$ Mère.

Par exemple, un langage dérivé de $\mathcal{AL}$ et contenant l'union, le prédicat existentiel et les restrictions de cardinalités s'appelera donc $\mathcal{ALUEN}$. L'union pouvant s'exprimer à l'aide du complément et de l'intersection ($C \sqcup D \equiv \neg(\neg C \sqcap \neg D)$) et le prédicat existentiel pouvant s'exprimer à l'aide du complément et du prédicat universel ($\exists R.C \equiv \neg\forall R.\neg C$), les constructions $\mathcal{U}$ et $\mathcal{E}$ sont facultatives si le complément est présent, et inversément. Le langage $\mathcal{ALUEN}$ peut donc être remplacé par le langage $\mathcal{ALCN}$ sans perte d'expressivité.

Un avantage important des connaissances exprimées en *Description Logics* est la possibilité de les utiliser avec des moteurs d'inférence afin d'obtenir davantage de connaissances. Ce qui importe principalement pour l'inférence de nouvelles connaissances est que la procédure se termine toujours en un temps fini mais ce temps peut varier selon le degré d'expressivité du langage de *Description Logic* utilisé. Les tâches d'inférence principales sont la vérification de satisfaisabilité d'une classe (est-il possible de trouver des individus répondant à toutes les conditions pour appartenir à cette classe ?), la détermination si une classe est sous-classe d'une autre en fonction des conditions d'appartenance à ces classes (ceci permet de produire automatiquement la hiérarchie d'héritage ou de compléter une hiérarchie existante), la vérification de la cohérence de classification des individus (les individus déclarés comme appartenant à une classe répondent-ils bien aux conditions d'appartenance à cette classe ?).

Le lecteur intéressé trouvera d'avantage d'informations sur les *Description Logics* dans [Baader et al., 2003].

2.4.2 OWL

OWL [McGuinness and van Harmelen, 2004] [Smith et al., 2004] est un langage de représentation d'ontologies développé par le W3C[3]. Ce langage a pour but de faciliter la compréhension d'informations par des machines et non simplement par des humains. Ceci est possible car en OWL les données sont représentées accompagnées de leurs structure et sémantique. OWL a été principalement conçu pour une utilisation dans le Semantic Web. Cette approche consiste à annoter chaque concept présent sur un site web à l'aide d'une référence à un concept dans une ontologie communément acceptée. Les recherches sur Internet pourraient alors se faire de manière sémantique (ou du moins via des requêtes plus riches que de simples mots-clés).

OWL se présente comme un langage XML basé sur RDF(S)-XML[4]. Il lui ajoute des constructions permettant de représenter des relations sémantiques plus complexes (ex : $\forall, \exists, \ldots$) venant principalement des Description Logics et qui seront détaillées par la suite.

OWL est composé de 3 sous-langages de richesse croissante en constructions :

OWL Lite possède une expressivité relativement limitée mais permet de facilement représenter des taxonomies de concepts et propriétés. Les cardinalités autorisées sont limitées à 0 et 1.

OWL DL permet d'exprimer presque toutes les expressions valides de la Description Logic

[3] `http://www.w3.org/2001/sw/WebOnt/`
[4] `http://www.w3.org/RDF/`

$\mathcal{SHOIN}$ et garantit une complétude des opérations d'inférence qui pourraient être effectuées sur des ontologies exprimées dans ce sous-langage.

OWL Full offre davantage encore d'expressivité (ex : une classe peut également être considérée comme une instance) mais ne garantit pas la complétude des opérations d'inférence.

Une description plus détaillée de contenu de ces différents langages et de leurs différences peut être trouvée dans [McGuinness and van Harmelen, 2004, Smith et al., 2004].

Dans notre méthode d'intégration, nous aurons besoin d'utiliser des moteurs d'inférence sur les ontologies des sites web pour rechercher les individus identiques. C'est pourquoi nous représenterons ces ontologies au moyen d'OWL-DL.

OWL DL se base sur la *Description Logic* $\mathcal{SHOIN}(\mathbf{D})$[Horrocks et al., 2003]. Ce langage fait partie de la famille $\mathcal{SH}$ ($\mathcal{S}$ à cause de ses relations avec la logique $S4_{(}m)$ et $\mathcal{H}$ pour Hiérarchie) qui est identique au langage $\mathcal{ALC}$ (cf. section 2.4.1) auquel on aurait ajouté la possibilité de rendre des *propriétés transitives* et la possibilité d'*organiser les propriétés selon une hiérarchie.*

$\mathcal{O}$ signifie que l'on a ajouté un constructeur permettant de définir une classe par extension (ex : Benelux ≡ {Belgique, Pays-Bas, Luxembourg}).

$\mathcal{I}$ signifie que l'on a ajouté une construction permettant de spécifier qu'une propriété est l'inverse d'une autre.

D signifie que l'on peut aussi utiliser des types de données simples (string,integer, ...) et des valeurs ("test", "42",...) là où on pouvait utiliser respectivement des classes et individus. Chaque type de données est en réalité considéré comme une classe dont les instances sont des valeurs littérales. Ceci a mené à distinguer en OWL deux types de propriétés (pour des raisons de réduction de la complexité de raisonnement) :

Les **ObjectProperties** permettent de représenter des relations entre classes. La valeur d'une telle propriété pour un individu sera donc toujours un autre individu. Elles pourraient donc être comparées aux associations dans le modèle *Entités-Associations* (cf. Annexe A).

Les **DatatypeProperties** quant à elles permettent de lier les individus d'une classe à des données de type simple (un nombre, une chaîne de caractères). Elles peuvent donc être comparées aux attributs dans le modèle *Entités-Associations* (cf. Annexe A).

OWL étant issu d'XML, permet d'inclure dans la *Description Logic* le mécanisme de *namespaces*[5]. Les namespaces sont des URI communes à un ensemble d'éléments. Par exemple dans `http://mondomaine.com/mydirectory#myElement`, `http://mondomaine.com/mydirectory#` est le namespace et `myElement` est un élément du namespace. Un ensemble de préfixes peut être associé à chaque namespace afin d'éviter l'utilisation des URI. Si le namespace précédent est remplacé par le préfixe *myns*, l'élément *myElement* pourra être utilisé de la manière suivante *myns :myElement.*

OWL permet via ce mécanisme d'importer d'autres ontologies dans une ontologie. Les structures de l'ontologie importée peuvent donc être utilisées dans l'ontologie courante comme si elles y étaient explicitement définies.

OWL permet également de définir différentes versions d'une ontologie, ainsi que de spécifier si la nouvelle version est compatible arrière ou non avec la structure de l'ancienne version de l'on-

[5] `http://www.w3.org/TR/REC-xml-names/`

tologie (si n'importe quel individu valide de l'ancienne version peut être considéré comme valide dans la nouvelle version). Il est également possible de spécifier qu'un composant de l'ontologie est périmé (*deprecated*), que son usage est donc déconseillé et qu'il n'est présent que pour assurer la validité des individus des anciennes ontologies dans la nouvelle.

Nous allons maintenant présenter les différentes constructions disponibles en OWL DL. Afin de faciliter la compréhension de l'utilité et du rôle des principales constructions, celles-ci seront présentées, accompagnées de leur constructeur correspondant en Description Logic et en logique des prédicats. Elles sont également illustrées sur un exemple. Il est à noter que les exemples doivent être considérés isolément et qu'ils n'ont aucun lien entre eux.

Construction	**Logique des prédicats**	**Description logic**	**OWL**
Concept Universel	$True$	$\top$	`owl:Thing`
Concept vide	$False$	$\bot$	`owl:Nothing`

Définition d'une classe	Personne(x) $\Leftrightarrow$ description	Personne$\equiv$ *description*	`<owl:Class rdf:ID="Personne">`
			`  <!-- description -->`
			`</owl:Class>`

Définition d'une sous-classe	$\forall$x.Femme(x) $\Rightarrow$ Personne(x)	Femme$\sqsubseteq$Personne	`<owl:Class rdf:ID="Femme">`
			`  <rdfs:subClassOf>`
			`  <owl:Class rdf:about="#Personne"/>`
			`  </rdfs:subClassOf>`
			`</owl:Class>`

Définition d'une propriété	Définition soit par extension (un ensemble de faits de type aPourMère(Pol,Christiane))
	
	ou par combinaison de propriétés déjà définies $\forall$x,y.aPourMère(x,y) $\Leftrightarrow$ aPourParent(x,y) $\wedge$ Femme(y)
		Pas de définition explicite	`<owl:ObjectProperty rdf:about="#aPourMere">`
			`  <rdfs:range rdf:resource="#Femme"/>`
			`  <rdfs:domain rdf:resource="#Personne"/>`
			`</owl:ObjectProperty>`

Définition d'une sous-propriété	$\forall$x,y.aPourMère(x,y) $\Rightarrow$ aPourParent(x,y)	aPourMère $\sqsubseteq$ aPourParent	`<owl:ObjectProperty rdf:ID="aPourMere">`
			`  <rdfs:subPropertyOf`
			`  rdf:resource="#aPourParent" />`
			`  ...`
			`</owl:ObjectProperty>`

Définition d'une classe équivalente	$\forall$x.Femme(x) $\Leftrightarrow$ Personne(x) $\wedge$ SexeFéminin(x)	Femme$\equiv$ Personne $\sqcap$ SexeFéminin	`<owl:Class rdf:ID="Femme">`
			`  <owl:equivalentClass>`
			`  <owl:Class>`
			`  <owl:intersectionOf`
			`  rdf:parseType="Collection">`
			`  <owl:Class rdf:about="#Personne"/>`
			`  <owl:Class rdf:about="#SexeFeminin"/>`
			`  </owl:intersectionOf>`
			`  </owl:Class>`
			`  </owl:equivalentClass>`
			`</owl:Class>`

Construction	**Logique des prédicats**	**Description logic**	**OWL**
Définition d'une propriété équivalente	∀x,y.hasMother(x,y) ⇔ aPourMère(x,y)	hasMother ≡ aPourMère	`<owl:ObjectProperty rdf:ID="hasMother"> <owl:equivalentProperty> <owl:ObjectProperty rdf:about="#aPourMere"/> </owl:equivalentProperty> </owl:ObjectProperty>`
Quantificateur Existentiel	∀x.Père(x) ⇔ Personne(x) ∧ SexeMasculin(x) ∧∃y.(estParentDe(x,y) ∧ Personne(y))	Père ≡ Personne ⊓ SexeMasculin ⊓ ∃estParentDe. Personne	`<owl:Class rdf:ID="Pere"> <owl:equivalentClass> <owl:Class> <owl:intersectionOf rdf:parseType="Collection"> <owl:Restriction> <owl:onProperty rdf:resource="#estParentDe"/> <owl:someValuesFrom rdf:resource="#Personne"/> </owl:Restriction> <owl:Class rdf:about="#Personne"/> <owl:Class rdf:about="#SexeMasculin"/> </owl:intersectionOf> </owl:Class> </owl:equivalentClass> </owl:Class>`
Quantificateur universel	∀x.PèreDeFilles(x) ⇔ Père(x) ∧(∀y.estParentDe(x,y) ⇒ Femme(y))	PèreDeFilles ≡ Père ⇒ ∀estParentDe.Femme	`<owl:Class rdf:ID="PereDeFilles"> <owl:equivalentClass> <owl:Class> <owl:intersectionOf rdf:parseType="Collection"> <owl:Restriction> <owl:onProperty rdf:resource="#estParentDe"/> <owl:allValuesFrom rdf:resource="#Femme"/> </owl:Restriction> <owl:Class rdf:about="#Pere"/> </owl:intersectionOf> </owl:Class> </owl:equivalentClass> </owl:Class>`
Intersection	∀x.Femme(x) ⇔ Personne(x) ∧ SexeFéminin(x)	Femme ≡ Personne ⊓ SexeFéminin	`<owl:Class rdf:about="#Femme"> <owl:intersectionOf rdf:parseType="Collection"> <owl:Class rdf:about="#Personne" /> <owl:Class rdf:about="#SexeFeminin" /> </owl:intersectionOf> </owl:Class>`
Union	∀x.Parent(x) ⇔ Père(x) ∨ Mère(x)	Parent ≡ Père ⊔ Mère	`<owl:Class rdf:about="#Parent"> <owl:unionOf rdf:parseType="Collection"> <owl:Class rdf:about="#Mere" /> <owl:Class rdf:about="#Pere" /> </owl:unionOf> </owl:Class>`

Construction	**Logique des prédicats**	**Description logic**	**OWL**

Complément	$\forall x.\text{Père}(x) \Leftrightarrow \neg\ \text{SexeFéminin}(x) \wedge \text{Parent}(x)$	$\text{Père} \equiv \neg\ \text{SexeFéminin} \sqcap \text{Parent}$	`<owl:Class rdf:ID="Pere">`
			`<owl:equivalentClass>`
			`<owl:Class>`
			`<owl:intersectionOf`
			`rdf:parseType="Collection">`
			`<owl:Class>`
			`<owl:complementOf`
			`rdf:resource="#SexeFeminin"/>`
			`</owl:Class>`
			`<owl:Class rdf:about="#Parent"/>`
			`</owl:intersectionOf>`
			`</owl:Class>`
			`</owl:equivalentClass>`
			`<rdfs:subClassOf rdf:resource="&owl;Thing"/>`
			`</owl:Class>`

Cardinalité minimale	$\forall x.\text{MèreFamNombreuse}(x) \Leftrightarrow \text{Mère}(x) \wedge \exists y_1,y_2,y_3.(y_1 \neq y_2 \wedge y_1 \neq y_3 \wedge y_2 \neq y_3 \wedge \text{estParentDe}(x,y_1) \wedge \text{estParentDe}(x,y_2) \wedge \text{estParentDe}(x,y_3))$	$\text{MèreFamNombreuse} \equiv \text{Mère} \sqcap \geq 3\text{estParentDe}$	`<owl:Class rdf:about="#MereFamNombreuse">`
			`<owl:intersectionOf rdf:parseType="Collection">`
			`<owl:Class rdf:about="#Mere" />`
			`<owl:Class>`
			`<rdfs:subClassOf>`
			`<owl:Restriction>`
			`<owl:onProperty`
			`rdf:resource="#estParentDe"/>`
			`<owl:minCardinality`
			`rdf:datatype="xsd:nonNegativeInteger">`
			`3`
			`</owl:minCardinality>`
			`</owl:Restriction>`
			`</rdfs:subClassOf>`
			`</owl:Class>`
			`</owl:intersectionOf>`
			`</owl:Class>`

Cardinalité maximale	$\forall x.\text{MèreDePetiteFam}(x) \Leftrightarrow \text{Mère}(x) \wedge \forall y_1,y_2,y_3,y_4.(y_1 \neq y_2 \wedge y_1 \neq y_3 \wedge y_1 \neq y_4 \wedge y_2 \neq y_3 \wedge y_2 \neq y_4 \wedge y_3 \neq y_4 \Rightarrow (\neg\ \text{estParentDe}(x,y_1) \vee \neg\ \text{estParentDe}(x,y_2) \vee \neg\ \text{estParentDe}(x,y_3) \vee \neg\ \text{estParentDe}(x,y_4)))$	$\text{MèreDePetiteFam} \equiv \text{Mère} \sqcap \leq 3\text{estParentDe}$	`<owl:Class rdf:about="#MereDePetiteFam">`
			`<owl:intersectionOf rdf:parseType="Collection">`
			`<owl:Class rdf:about="#Mother" />`
			`<owl:Class>`
			`<rdfs:subClassOf>`
			`<owl:Restriction>`
			`<owl:onProperty`
			`rdf:resource="#estParentDe"/>`
			`<owl:maxCardinality`
			`rdf:datatype="xsd:nonNegativeInteger">`
			`3`
			`</owl:maxCardinality>`
			`</owl:Restriction>`
			`</rdfs:subClassOf>`
			`</owl:Class>`
			`</owl:intersectionOf>`
			`</owl:Class>`

Classe énumérée (décrite par extension)	$\forall x.\text{Benelux}(x) \Leftrightarrow x = \text{Belgique} \vee x = \text{Pays-Bas} \vee x = \text{Luxembourg}$	$\text{Benelux} \equiv \{\text{Belgique, Pays-Bas, Luxembourg}\}$	`<owl:Class rdf:ID="Benelux">`
			`<owl:equivalentClass>`
			`<owl:Class>`
			`<owl:oneOf rdf:parseType="Collection">`
			`<rdf:Description rdf:about="#Belgique"/>`
			`<rdf:Description rdf:about="#Luxembourg"/>`
			`<rdf:Description rdf:about="#Pays-Bas"/>`
			`</owl:oneOf>`
			`</owl:Class>`
			`</owl:equivalentClass>`
			`</owl:Class>`

Construction	Logique des prédicats	Description logic	OWL

Définition de la propriété inverse d'une propriété	∀x,y.aPourMère(x,y) ⇔ estMèreDe(y,x)	aPourMère ≡ estMèreDe $^-$	`<owl:ObjectProperty rdf:ID="aPourMere">`
			`<rdfs:domain rdf:resource="#Personne"/>`
			`<rdfs:range rdf:resource="#Femme"/>`
			`<owl:inverseOf rdf:resource="#estMereDe"/>`
			`</owl:ObjectProperty>`
			`<owl:ObjectProperty rdf:ID="estMereDe">`
			`<rdfs:domain rdf:resource="#Femme"/>`
			`<rdfs:range rdf:resource="#Personne"/>`
			`<owl:inverseOf rdf:resource="#aPourMere"/>`
			`</owl:ObjectProperty>`

Définition d'une propriété transitive	∀x,y,z.estAncêtre(x,y) ∧ estAncêtre(y,z) ⇒ estAncêtre(x,z)	Tr(estAncêtreDe)	`<owl:ObjectProperty rdf:ID="estAncêtreDe">`
			`<rdf:type`
			`rdf:resource="owl:TransitiveProperty"/>`
			`<rdfs:domain rdf:resource="#Personne"/>`
			`<rdfs:range rdf:resource="#Personne"/>`
			`</owl:ObjectProperty>`

Définition d'une propriété symétrique	∀x,y.estConjointDe(x,y) ⇔ estConjointDe(y,x)	estConjointDe ≡ estConjointDe$^-$	`<owl:ObjectProperty rdf:ID="estConjointDe">`
			`<rdf:type`
			`rdf:resource="owl:SymmetricProperty"/>`
			`<rdfs:domain rdf:resource="#Personne"/>`
			`<rdfs:range rdf:resource="#Personne"/>`
			`<owl:inverseOf rdf:resource="#estConjointDe"/>`
			`</owl:ObjectProperty>`

Définition d'une propriété fonctionnelle	∀x.∀y_1,y_2.($y_1 \neq y_2 \Rightarrow$ (¬ estParentDe(x,y_1) ∨ ¬ estParentDe(x,y_2)))	⊤ ⊑≤1 aPourMère	`<owl:ObjectProperty rdf:ID="aPourMere">`
			`<rdf:type`
			`rdf:resource="owl:FunctionalProperty"/>`
			`<rdfs:domain rdf:resource="#Personne"/>`
			`<rdfs:range rdf:resource="#Femme"/>`
			`<owl:inverseOf rdf:resource="#estMereDe"/>`
			`</owl:ObjectProperty>`

Définition d'une propriété fonctionnelle inverse	∀x.∀y_1,y_2.($y_1 \neq y_2 \Rightarrow$ (¬ estMèreDe(x,y_1) ∨ ¬ estMèreDe(x,y_2))) aPourMère(x,y) ⇔ estMèreDe(y,x)	⊤ ⊑≤1 estMèreDe$^-$	`<owl:ObjectProperty rdf:ID="estMereDe">`
			`<rdf:type rdf:resource=`
			`"owl:InverseFunctionalProperty"/>`
			`<rdfs:domain rdf:resource="#Femme"/>`
			`<rdfs:range rdf:resource="#Personne"/>`
			`<owl:inverseOf rdf:resource="#aPourMere"/>`
			`</owl:ObjectProperty>`

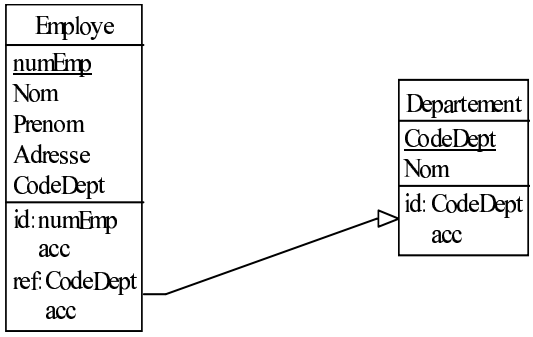
TAB. 2.3: Correspondances entre la logique des prédicats, la Description Logic $\mathcal{SHOIN}$ et les constructions OWL

Les notions de sous-classes et de classes équivalentes bien qu'apparentées sont différentes. Si on reprend l'exemple utilisé pour les sous-classes dans le précédent tableau, cela signifie que l'ensemble des femmes est un sous-ensemble des personnes, mais le fait d'être une personne ne suffit pas pour être une femme. Il s'agit donc d'une condition nécessaire mais non suffisante. Par contre dans l'exemple sur les classes équivalentes, le fait d'être une personne de sexe féminin suffit (et est nécessaire) à être une femme. Il s'agit donc d'une condition nécessaire et suffisante. La relation d'équivalence entre classes peut être vue comme une relation de sous-classe mutuelle entre les deux classes (chacune étant un sous-ensemble de l'autre, elles représentent donc des ensembles identiques).

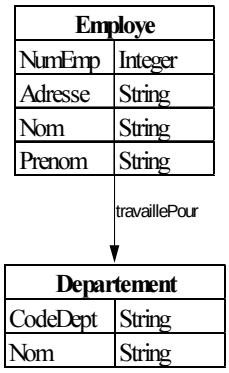
2.4.3 SPARQL

SPARQL [Prud'hommeaux and Seaborne, 2004] est un langage de syntaxe proche du SQL permettant d'exécuter des requêtes sur des graphes RDF [6]. L'OWL se basant sur RDF, ces requêtes peuvent également être utilisées pour ce langage d'ontologies.

Base de données relationnelle



Ontologie OWL



Requête

Récupérer les noms, prénoms et adresses des employés travaillant dans le département de comptabilité (code : COMPTA)

SQL

```
SELECT e.Nom,e.Prenom,e.Adresse
FROM Employe e, Departement d
WHERE e.CodeDept = d.CodeDept AND
      e.CodeDept = 'COMPTA'
```

SPARQL

```
PREFIX model:
<http://localhost/owl/SPARQLExample.owl\#>
SELECT ?Nom ?Prenom ?Adresse
WHERE { ?emp model:travaillePour ?dept.
 ?dept model:CodeDept 'COMPTA' .
 ?emp model:Nom ?Nom .
 ?emp model:Prenom ?Prenom .
 ?emp model:Adresse ?Adresse .
     }
```

TAB. 2.4 – Exemple de correspondance entre une requête SQL et une requête SPARQL effectuée sur une base de données relationnelle et une ontologie OWL de même contenu informationnel.

La partie haute du tableau 2.4 présente une base de données relationnelle et une ontologie OWL contenant toutes deux des informations sur des employés travaillant dans des départements. La partie basse du tableau quant à elle montre la correspondance entre une requête SQL (exécutée sur la base de données) et requête SPARQL (exécutée sur l'ontologie) permettant toutes deux de donner les noms,prénoms et adresses des employés du département de comptabilité.

Les requêtes SPARQL fonctionnent à l'aide de variables (?Nom, ?Prénom, ?Adresse, ?emp, ?dept dans l'exemple) et de triplets RDF (*domaine propriété cible*). Les variables présentes dans une requête peuvent représenter n'importe quel élément d'une ontologie (un individu, une classe, une propriété).

[6] les ontologies RDF sont souvent représentées comme des graphes dans lesquels les classes sont des noeuds et les propriétés sont des arcs

Lors de l'exécution, le moteur de requêtes SPARQL tente de trouver des valeurs satisfaisant aux différentes conditions pour les différentes variables. Par exemple *?emp model :travaillePour ?dept* signifie que l'objet de l'ontologie remplaçant la variable *?emp* doit posséder une propriété nommée *model :travaillePour* ayant pour valeur l'objet remplaçant la variable *?dept*. Dans ce cas précis cela signifie que l'employé représenté par *?emp* travaille dans le département représenté par *?dept*, ce qui correspond aux jointures naturelles en SQL.

La ligne `PREFIX model: <http://localhost/owl/SPARQLExample.owl#>` permet de remplacer dans la suite de la requête le namespace `http://localhost/owl/SPARQLExample.owl#` par le préfixe "model". `model:travaillePour` remplace donc `http://localhost/owl/SPARQLExample.owl#travaillePour`.

2.4.4 Protégé

Protégé[7] est un éditeur et visualisateur d'ontologies open-source. Celui-ci possède une architecture évolutive par l'intermédiaire de divers plugins. L'éditeur est à l'origine construit sur la base d'un modèle de connaissance de type frame-based [Noy et al., 2000] (classes, slots, instances). Par la suite un plugin a été implémenté par- dessus ce modèle afin de pouvoir éditer et visualiser des ontologies OWL. L'outil offre également une API permettant de manipuler des ontologies OWL ou frame-based sans devoir passer par l'interface du logiciel, ceci permettant de réaliser des outils externes manipulant ces ontologies.

Dans le modèle frame-based, les propriétés sont nommées slots et les individus sont nommés instances. Contrairement aux ontologies telles que présentées à la section 2.2, deux types de slots sont à distinguer, les *own slots* et les *template slots*.

Un own-slot est attaché à un objet d'une base de connaissances (classe, propriété, instance) et décrit un attribut propre à cet objet. Les own-slots d'une classe ou d'une propriété ne sont pas hérités par les sous-classes/propriétés ni par les instances contrairement aux template-slots.

Un template slot qui ne peut être attaché qu'à une classe, est hérité par les sous-classes et est matérialisé par un *own-slot* pour les instances de cette classe.

[7] `http://protege.stanford.edu`

Chapitre 3

L'intégration d'ontologies

Etant donné la diversité des documents pouvant entrer dans l'état de l'art du processus d'intégration, nous commencerons dans ce chapitre par présenter l'intégration de schémas et d'instances au travers de principes généraux et en nous appuyant sur des articles d'auteurs renommés. Par la suite, afin d'approfondir un peu la problématique, nous présenterons un état de l'art des outils les plus récents pouvant s'intégrer dans la problématique de l'intégration de schémas ou d'ontologies.

3.1 Intégration de schémas

3.1.1 Définition de l'intégration

En généralisant la définition orientée bases de données proposée par S. Spaccapietra et C. Parent dans [Parent and Spaccapietra, 2000] ,nous obtenons la définition suivante :

L'intégration est le processus qui, prenant en entrée un ensemble de bases de données, schémas ou ontologies, produit comme résultat une représentation unifiée des entrées ainsi qu'une liste des correspondances existant entre ces différentes entrées et permettant l'accès à ces dernières via le schéma global.

Selon J-L Hainaut dans [Hainaut, 2002] concernant l'intégration de schémas conceptuels :

"(L'intégration consiste à) produire, étant donné N schémas S_i *relatifs à des portions d'un même domaine d'application, un schéma global unique* S *qui reprend les concepts corrects de tous ces schémas. Tout type de fait représenté dans un schéma* S_i *est aussi présent dans* S*. Tout type de fait représenté dans* S *l'est aussi dans au moins l'un des* S_i*."*

L'intégration consiste donc à produire à partir de N schémas distincts un schéma unique contenant l'ensemble de l'information représentée dans les N schémas.

3.1.2 Processus général d'intégration

Dans [Parent and Spaccapietra, 2000] et [Spaccapietra, 2005] les auteurs distinguent différentes étapes qui sont nécessaires lors de la production d'une représentation globale quelle que soit la forme des entrées.

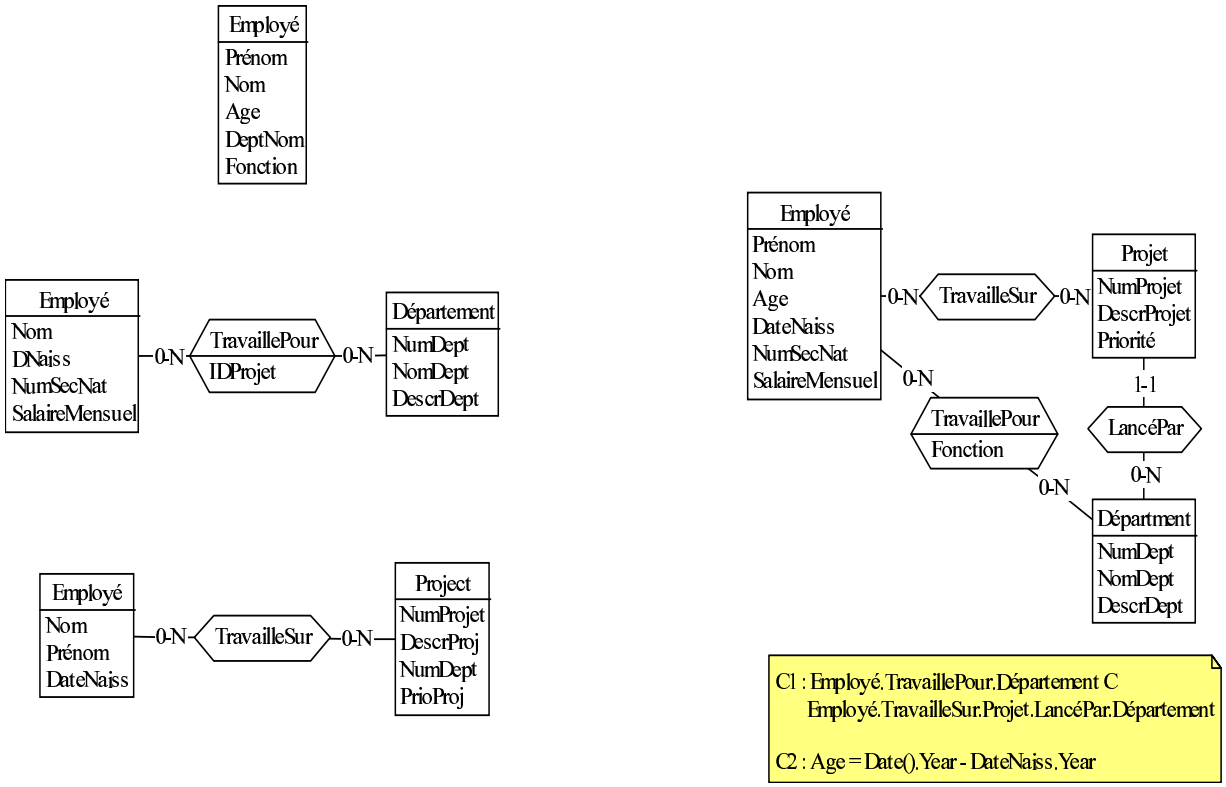


FIG. 3.1 – Exemple d'intégration de schémas

Ces étapes sont les suivantes :

- La pré-intégration
  - L'homogénéisation
  - L'enrichissement sémantique
- La recherche des correspondances
- L'identification des conflits dans les données
- L'intégration proprement dite

Pré-Intégration

Homogénéisation La nécessité d'homogénéiser les schémas fournis en entrée provient de leurs différentes origines.

Dans sa vue la plus générale, le processus d'intégration peut être effectué à partir de schémas dans des formalismes différents (ex : intégration d'un schéma ERA, d'une ontologie et d'un schéma relationnel). Ces différents modèles proposent en général des constructions différentes, qui possèdent parfois des niveaux de précision relativement différents. Par exemple, le modèle ERA propose une distinction entre types d'entités et types d'associations alors que dans le modèle relationnel, l'une comme l'autre sont représentées comme des relations. L'homogénéisation consisterait donc à pouvoir distinguer les relations représentant des entités de celles représentant des associations. Cette partie du processus est également connu sous le nom de rétro-ingénierie.

L'hétérogénéité des schémas ne provient pas exclusivement du formalisme utilisé pour les représenter mais peut également venir de leur concepteur. En effet, deux personnes modélisant le même domaine à l'aide du même formalisme aboutiront en général à des schémas différents.

L'homogénéisation de schémas consiste donc à choisir un modèle de représentation commun pour l'ensemble des schémas de départ appelé modèle pivot. Ce choix est loin d'être évident. De nombreux chercheurs proposent d'utiliser comme pivot le modèle orienté objet. Cependant plus un modèle est riche en constructions différentes, plus il existe de façons différentes de représenter le même fait (phénomène connu sous le nom de relativisme sémantique). Afin de rendre le processus d'intégration plus simple, le choix devra se porter sur un modèle proposant peu de constructions, offrant une seule façon de représenter chaque fait mais permettant une expressivité suffisante. Une autre possibilité est de contraindre le concepteur à favoriser certaines constructions via des règles organisationnelles.

Enrichissement sémantique Un autre problème auquel on doit faire face lorsqu'on veut intégrer des schémas (quelle que soit la richesse du modèle dans lequel ils sont exprimés) est l'absence de représentation sur le schéma de toutes les contraintes réellement en vigueur sur les données qu'ils contiennent. En effet, de nombreuses contraintes sont implicites car connues de tous les membres de l'organisation à laquelle appartient le schéma ou parce qu'elles sont appliquées au travers de programmes d'application accédant aux données. Une intervention humaine est donc nécessaire pour compléter la sémantique présente sur le schéma afin de savoir ce que représente réellement chaque concept de celui-ci et permettre ainsi une intégration correcte.

Recherche des correspondances

Cette étape parfois aussi appelée *matching* consiste en un processus recevant en entrée deux schémas ou ontologies et produisant comme résultat la liste des correspondances (ou mappings) entre leurs éléments (ex : l'élément "livre" d'un schéma correspond à l'élément "ouvrage" de l'autre schéma). Cette opération est de toute première importance dans de nombreux domaines comme le Web Sémantique, l'intégration de schémas et les datawarehouses. Elle sera développée davantage à la section 3.1.5.

Intégration proprement dite

Lors de cette phase, les correspondances (ou mappings) découvertes lors de la phase de *recherche des correspondances* sont utilisées afin de d'unifier les différents schémas en un seul.

Bien que constituant la suite logique des précédentes étapes, il n'existe pas un seul bon schéma global ni une seule bonne façon de produire ce schéma. En réalité le schéma final dépendra beaucoup des buts pour lesquels il a été conçu et des règles qui ont été imposées pour sa réalisation.

Dans [Parent and Spaccapietra, 2000] les auteurs isolent 3 types de buts conduisant à des schémas relativement différents. Ces buts sont :

- La simplicité/lisibilité
- La complétude
- L'exhaustivité

La simplicité : le but ici est de produire un schéma avec le minimum d'éléments afin de garder un schéma relativement petit, aisément lisible et compréhensible par un utilisateur. Pour ce faire, si deux concepts (provenant de schémas différents) sont comparables, ils seront représentés par un unique concept dans le schéma global.

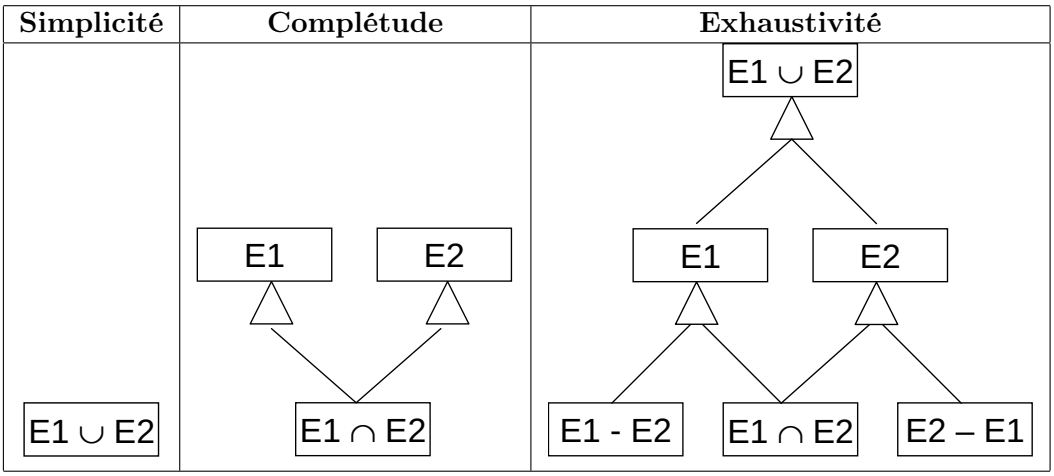


FIG. 3.2 – Représentation visuelle des buts d'intégration présenté par S. Spaccapietra (E1 ∩ E2 ≠ ϕ)

La complétude **:** le but ici est de conserver explicitement les différences existant dans les schémas sources. Ceci permet de retrouver facilmentla provenance des composants du schéma global, pour pouvoir répercuter sur celui-ci les changements effectués sur les schémas sources. Si des concepts des différents schémas possèdent une intersection non vide, les concepts seront distincts et un sous-type de ces concepts sera ajouté afin de représenter leur intersection.

L'exhaustivité **:** alors que les deux précédentes approches représentaient uniquement les faits présents dans les schémas d'origine, celle-ci propose de compléter ces faits afin de refléter toutes les combinaisons possibles entre ceux-ci. Cette approche permet d'anticiper et par conséquent de faciliter l'ajout de nouveaux schémas. Dans notre exemple, les différents concepts seront représentés, mais aussi leur union, leur intersection et leur différence.

3.1.3 Stratégies d'intégration

Etant donné l'ampleur de la tâche lorsque de nombreux schémas doivent être intégrés, des stratégies doivent être mises au point afin de réaliser la tâche le plus efficacement et précisément possible. Dans [Hainaut, 2002], [Parent and Spaccapietra, 2000] et [Spaccapietra, 2005], les auteurs distinguent trois stratégies principales, à savoir la stratégie *one shot*, la stratégie *hiérarchique* et la stratégie *incrémentale*.

Stratégie one shot

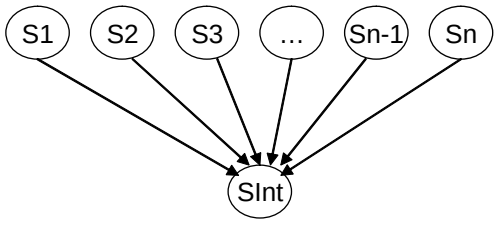


FIG. 3.3 – Schéma d'intégration one shot

Dans la stratégie one shot, tous les schémas sources sont intégrés en une seule fois en un schéma unique. Ce type d'intégration requiert que les schémas soient relativement similaires ou du moins que certains concepts soit présents dans la plupart d'entre-eux. Le problème de cette approche est de ne pas pouvoir revenir en arrière étant donné l'absence d'étape intermédiaire. Une

fois l'intégration effectuée, le seul moyen de corriger une erreur ou de répercuter un changement des sources est de procéder à nouveau à l'intégration de l'ensemble. Ce type d'intégration pourrait s'appliquer lors du regroupement des bases de données des étudiants de différentes écoles, les informations mémorisées à propos des étudiants étant généralement similaires. Cependant il faut que le nombre d'écoles soit relativement limité pour que cette stratégie puisse être employée.

Stratégie hiérarchique

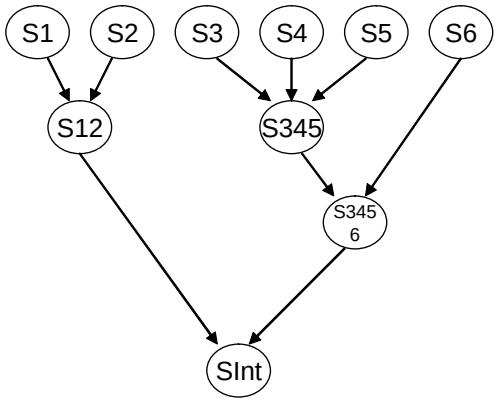


FIG. 3.4 – Schéma d'intégration hiérarchique

Dans la stratégie hiérarchique, l'intégration s'effectue à plusieurs niveaux successifs. Des sous-ensembles de l'ensemble des schémas de départ sont intégrés. Les schémas globaux produits sont ensuite également intégrés par sous-ensembles et ainsi de suite jusqu'à obtenir un seul schéma, le schéma global final. Cette approche convient particulièrement bien lorsque les schémas sources sont naturellement décomposés en sous-systèmes hiérarchisés. C'est-à-dire lorsque des schémas peuvent être regroupés selon une thématique commune ou lorsque leur schéma global permet à lui seul de résoudre des problèmes qui n'auraient pas pu l'être à l'aide des schémas individuels (ex : le regroupement des schémas utilisés pour la gestion des stocks de matières premières, la gestion de stocks des produits finis et le contrôle du processus de production permet de gérer l'ensemble du processus de production).

Stratégie incrémentale

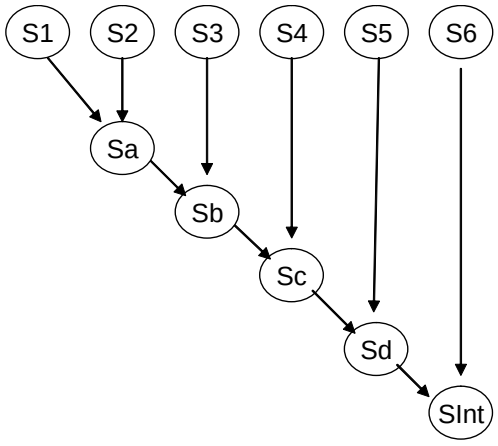


FIG. 3.5 – Schéma d'intégration incrémentale

Dans la stratégie incrémentale, on classe tout d'abord les schémas par ordre décroissant selon un critère tel que la fiabilité, la représentativité du domaine, ... On prend ensuite le schéma considéré comme le meilleur selon le critère choisi et on l'intègre avec le suivant. Le schéma global produit est ensuite intégré avec le 3^e schéma et ainsi de suite jusqu'à avoir un schéma global de l'ensemble. Cette approche permet de prendre en compte les situations où les schémas sont trop nombreux ou trop complexes pour permettre une intégration one shot, ainsi que pour les ensembles des schémas non organisés en sous-systèmes hiérarchisés, empêchant l'utilisation de la méthode hiérarchique.

Cette méthode permet d'accorder la priorité aux schémas considérés comme les plus pertinents. Au fur et à mesure de l'avancement dans le processus d'intégration, de moins en moins de concepts seront ajoutés au schéma global, la plupart étant déjà présents lors des phases précédentes. Cette méthode pourrait être employée pour intégrer de nombreux schémas traitant d'une même thématique. Etant donné leur nombre, il est presque impossible de produire en une seule fois un schéma global de bonne qualité. De plus, comme tous les schémas concernent une même thématique, on ne peut les grouper en sous-systèmes. Ceci s'adapterait donc bien à notre optique d'intégration de sites web.

3.1.4 Types d'architectures intégrées

Lorsqu'on possède un schéma global issu de l'intégration de divers schémas locaux, il faut ensuite s'intéresser au sort réservé aux données associées aux schémas locaux. En effet, celles-ci doivent également être accessibles à partir du schéma global. Pour ce faire, deux approches sont possibles. La première consiste à exporter l'ensemble des données locales vers le schéma global. La deuxième consiste à laisser les données dans les schémas locaux. La manipulation des données à partir du schéma global se fait alors à l'aide des correspondances repérées entre les schémas locaux et le schéma global.

L'approche par fusion

Dans l'approche par fusion, le schéma global n'a de relation avec les schémas locaux que lors de l'intégration. En effet, une fois le schéma global produit, les données associées aux différents schémas sources sont exportées dans un format compatible avec le schéma global et sont associées à celui-ci.

L'inconvénient de cette approche est qu'en cas de mise à jour des données, toutes doivent être supprimées et réexportées à partir des sources, aucun lien ne subsistant entre une donnée et sa source d'origine.

De plus, entre deux exportations, les données du schéma global ne sont plus synchronisées avec les sources et sont donc susceptibles de contenir des informations périmées ou erronées. La cohérence doit par conséquence être garantie par des exportations régulières. Ceci est peu concevable pour des masses importantes de données vu le temps nécessaire à l'exportation. Cependant, des exportations trop rapprochées par rapport à la fréquence de changement causeraient des transferts inutiles, puisque les données présentes dans la représentation intégrée sont toujours valables. Ce type d'approche est par contre utile si les sources ne doivent pas être conservées et que seul le schéma global doit subsister (ex : une base de données qui en remplace plusieurs autres). Cette approche est également envisageable dans le cas de l'intégration de sites web. Si les données ne changent pas trop fréquemment et que la récupération des données n'est pas trop longue ou si la possession de la dernière version des données n'est pas critique, les performances d'accès peuvent être grandement améliorées car les données intégrées peuvent être stockées plus près de l'utilisateur (ex : sur son serveur personnel).

Les approches par correspondances

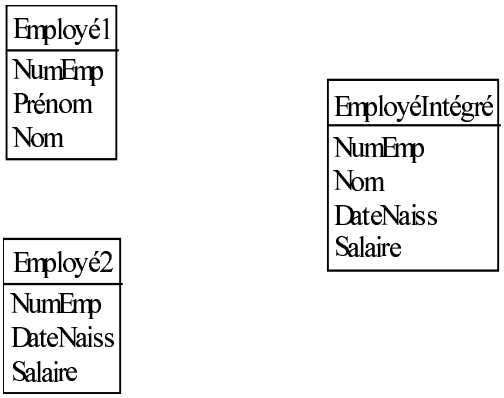
Dans cette seconde approche, les données ne sont pas stockées physiquement dans un format compatible avec le schéma global. En effet, ces dernières demeurent dans leur emplacement d'origine mais ceci est complètement transparent pour l'utilisateur effectuant des requêtes à partir du schéma global. Celles-ci sont répercutées sur les schémas locaux afin de récupérer ou répartir l'information nécessaire à leur exécution. Ceci est réalisé à l'aide de deux composants principaux.

- les *wrappers* qui traduisent les informations entre les structures du schéma global et des schémas sources.
- le *médiateur* qui contient le schéma global ainsi que les correspondances entre ce schéma et les schémas sources. C'est également le médiateur qui divise les requêtes en sous-requêtes destinées à chacune des sources, pour ensuite récupérer et combiner les résultats provenant de celles-ci.

Cette deuxième approche se décline en trois grandes tendances : les approches Global as View (GaV), Local as View (LaV) et mixte. Chacune possède des qualités et défauts particuliers qui seront examinés ci-dessous.

L'approche Global as View (GaV) Dans [Calvanese et al., 2001] les auteurs définissent formellement cette architecture de la manière suivante :

> *"We assume that we have a query language $\mathcal{V}_S$ over the alphabet $\mathcal{A}_f$ (the local ontologies' alphabet), and the mapping between the global and the local ontologies is given by associating to each term in the global ontology a* view*,i.e., a query over the local ontologies. The intended meaning of associating to the term C in the global schema ($\mathcal{G}$) a query $\mathcal{V}_f$ over $\mathcal{S}$ (the local ontologies), is that such a query represents the best way to characterize the instances of C using the concepts of $\mathcal{S}$"*




```
CREATE VIEW EmployéIntégré(NumEmp,Nom,DateNaiss,Salaire)
AS
SELECT e1.NumEmp,concat(e1.Prénom,e1.Nom),e2.DateNaiss,e2.Salaire
FROM Employé1 e1,Employé2 e2
WHERE e1.NumEmp = e2.NumEmp
```

FIG. 3.6 – Exemple d'une requête de définition de vue de type GaV

En d'autres termes, cette approche consiste à exprimer chaque concept du schéma global comme une vue (au sens bases de données) sur les schémas locaux. Le schéma global contient une description de l'ensemble des informations disponibles dans les différents schémas sources. Notons que dans des cas particuliers, des parties des schémas locaux peuvent ne pas être consi-

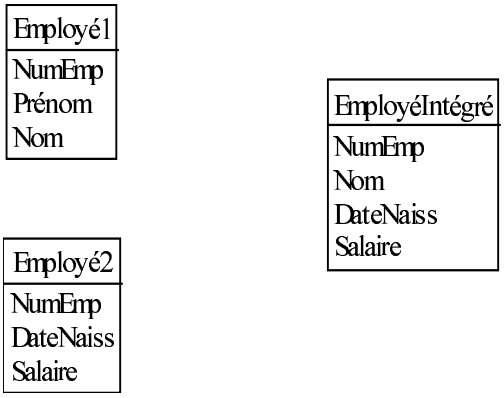
dérées comme importantes dans le schéma global et donc être laissées de côté.

Cette approche est souvent adoptée dans les systèmes d'intégration de bases de données, car elle permet une grande simplicité pour l'exécution de requêtes. Ces dernières doivent simplement être répercutées sur les bons schémas sources en fonction des requêtes définissant les vues. Un exemple de ces vues est donné à la figure 3.6.

Lorsqu'on veut ajouter un nouveau schéma source ou prendre en compte des informations supplémentaires ou des modifications dans les schémas sources, la seule manière de procéder est de revoir l'ensemble du schéma global ainsi que toutes les vues qui s'y rapportent. Dans [Calvanese et al., 2001], les auteurs démontrent également que lorsqu'on ajoute des contraintes (même simples) au schéma global, l'exécution des requêtes devient très rapidement problématique.

L'approche Local as View (LaV) A nouveau les auteurs de [Calvanese et al., 2001] proposent une définition formelle :

> *"We assume that we have a query language $\mathcal{V}_\mathcal{G}$ over the alphabet $\mathcal{A}_\mathcal{G}$ (the global ontology's alphabet), and the mapping between the global and the local ontologies is given by associating to each term in the local ontologies a* view*,i.e., a query over the global ontology. The intended meaning of associating to the term $\mathcal{C}$ in the set of local ontologies ($\mathcal{S}$) a query $\mathcal{V}_\mathcal{G}$ over $\mathcal{G}$ (the global ontology), is that such a query represents the best way to characterize the instances of $\mathcal{C}$ using the concepts of $\mathcal{G}$ "*




```
CREATE VIEW Employé1(NumEmp,Prénom,Nom)
AS
SELECT NumEmp,substring_before(' ',Prénom),substring_after(' ',Nom)
FROM EmployéIntégré
```

FIG. 3.7 – Exemple d'une requête de définition de vue de type LaV

En d'autres termes, à l'inverse du Global As View, cette approche consiste à exprimer les concepts de chaque schéma source comme une vue du schéma global, qui est lui spécifié indépendamment des sources. Celui-ci contient une représentation de l'ensemble des concepts présents dans les schémas locaux mais il peut également contenir d'autres concepts qui ne sont actuellement présents dans aucun des autres schémas.

Ce type d'architecture se prête très bien à une évolution fréquente des schémas sources ou à l'ajout de nouvelles sources. Ainsi tout le système ne doit pas être revu. Seules les vues concernant le schéma source en cause doivent être modifiées ou créées, le schéma global demeurant

inchangé.

Le principal problème de l'approche est que, contrairement à ce qui se produit dans l'approche GaV, on ne peut pas ici se baser sur les vues afin de découper automatiquement les requêtes en sous-requêtes . En effet, dans ce cas-ci, les vues ne fournissent pas d'information permettant de savoir facilement dans quelle source se situe un concept particulier du schéma global. Pour cela, il faut interroger chaque schéma local afin de voir s'il contient le concept en question. Une étape d'inférence est donc nécessaire afin de déduire la meilleure manière de répondre à la requête en combinant les différentes sources. Cette étape est relativement gourmande en temps de calcul et possède une complexité exponentielle.

L'approche Mixte Etant donné les désavantages des deux précédentes approches, l'idée de les combiner est apparue afin d'obtenir une architecture réunissant leurs avantages sans en posséder les inconvénients. Ceci est réalisé en ne contraignant pas le sens d'expression des correspondances.

Les auteurs de [Calvanese et al., 2001] définissent l'approche mixte de la manière suivante :

> In the unrestricted approach, we have both a query language $\mathcal{V}_\mathcal{S}$ over the alphabet $\mathcal{A}_\mathcal{S}$, and a query language $\mathcal{V}_\mathcal{G}$ over the alphabet $\mathcal{A}_\mathcal{G}$, and the mapping between the global and the local ontologies is given by relating views over the global ontology to views over the local ontologies. Again, the intended meaning of relating the view $\mathcal{V}_\}$ over the global ontology to the view Vs over the local ontology is that $\mathcal{V}_\int$ represents the best way to characterize the objects satisfying $\mathcal{V}_\}$ in terms of the concepts in $\mathcal{S}$.

Cependant, bien qu'étant la seule approche possédant un niveau d'expressivité suffisant pour couvrir tous les cas possibles, cette méthode est également la plus difficile à mettre en oeuvre car elle regroupe les difficultés des deux autres.

Certains auteurs tels que [Calì et al., 2002] n'utilisent pas une méthode permettant des correspondances bidirectionnelles pour leur approche mixte. En effet, les auteurs démontrent la possibilité de conversion de LaV en GaV sous certaines conditions tout en conservant des résultats de requêtes identiques. Ceci permet de bénéficier de l'évolutivité aisée du LaV tout en étant en mesure de convertir le système en GaV afin de fournir aisément les réponses aux requêtes qui pourraient être soumises au système.

	Exécution des requêtes	**Evolutivité**
GaV	+	-
LaV	-	+
Mixte	+	+

TAB. 3.1 – Tableau récapitulatif des + et - des approches GaV,LaV et mixte

3.1.5 Recherche des correspondances (Matching)

Comme mentionné à la section 3.1.2, le matching consiste en un processus recevant en entrée deux schémas ou ontologies et produisant comme résultat la liste des correspondances (ou mappings) entre ceux-ci. La recherche manuelle des correspondances s'avère relativement longue, sujette à erreur et nécessite une très bonne connaissance du domaine représenté de la part de la personne effectuant cette tâche. De nombreuses théories ont donc été développées par les chercheurs afin de tenter d'extraire les correspondances de la manière la plus automatique et la plus efficace possible. Ces approches sont toutes relativement différentes et se basent sur des informations diverses contenues dans les schémas ou ontologies pour choisir les paires de concepts similaires.

Dans cette section nous verrons tout d'abord que la notion de *correspondance* est relativement large et peut être divisée en sous-catégories. Ensuite, nous présenterons les principales catégories permettant de classer les méthodes de recherche de correspondances en nous basant sur les études comparatives [Rahm and Bernstein, 2001] et [Shvaiko and Euzenat, 2005].

Cardinalités des correspondances

Lorsqu'on effectue une recherche des correspondances entre deux ontologies ou schémas, toutes les correspondances ne peuvent être considérées comme équivalentes. En effet, certaines correspondances relient un composant d'une ontologie à un composant d'une autre ontologie (ex : un âge), d'autres relient plusieurs composants d'une ontologie à un seul composant de l'autre ontologie (ex : un nom ↔ un nom et un prénom). Ces deux types de correspondances se distinguent par des cardinalités de matching différentes, car ils ne relient pas le même nombre de composants. Cette classification peut encore être affinée en distinguant le type de composants entre lesquels les correspondances existent (une classe, une propriété, ...).

Dans [Rahm and Bernstein, 2001], les auteurs proposent une classification de ces cardinalités et distinguent les types suivants : les cardinalités au niveau du schéma et les cardinalités au niveau des instances. Les cardinalités au niveau du schéma sont alors subdivisées en deux sous-catégories : le niveau structurel et le niveau des éléments. Tous deux sont subdivisés en : cardinalités locales et globales. Ces différents types de cardinalités sont détaillés ci-dessous et un résumé de leur classification est présenté à la figure 3.8 alors qu'un exemple est fourni à la figure 3.9 et dans le tableau 3.2.

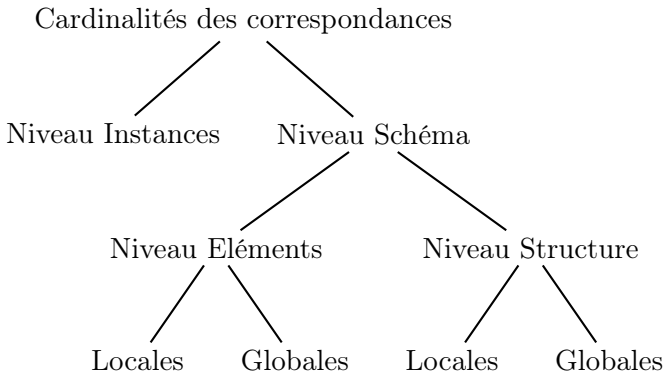


FIG. 3.8 – Les différents types de cardinalités de matching

Les cardinalités de correspondances :

- *Les cardinalités au niveau des instances* désignent le nombre d'instances d'un schéma auxquelles une instance de l'autre schéma est reliée via les relations de correspondance entre les concepts auxquels elles appartiennent.
- *Les cardinalités au niveau des éléments* concernent uniquement les éléments de niveau atomique (non encore décomposables) ou décidés comme tels (ex : les attributs atomiques dans le modèle ERA)
- *Les cardinalités au niveau de la structure* concernent des groupes d'éléments atomiques participant (ou non) dans leur ensemble à des relations de correspondance.
- *Les cardinalités locales* désignent le nombre de concepts (ou groupe de concepts) d'un schéma et le nombre de concepts (ou groupes de concepts) de l'autre schéma associés dans **1** relation de correspondance.
- *Les cardinalités globales* désignent le nombre de relations de correspondance auxquelles un concept ou groupe de concepts participe.

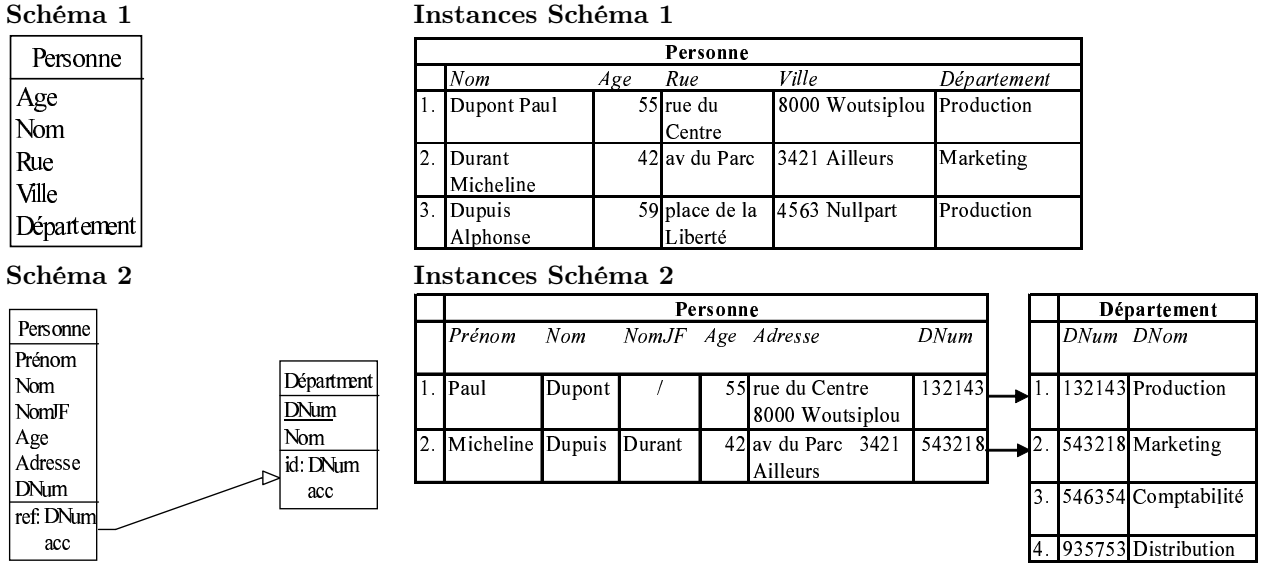


FIG. 3.9 – Exemples de deux schémas représentant des personnes travaillant dans des départements.

Schéma 1	**Schéma 2**	**Niveau Schéma**		**Niveau Instances**
		Niveau Eléments - Locales	**Niveau Structure - Locales**	
Personne.Age	Personne.Age	1 :1	1 :1	1 :1
Personne.Nom	Personne.Prénom ; Personne.Nom	1 :n	1 :1	1 :1
	Personne.Prénom ; Personne.NomJF	1 :n	1 :1	1 :1
Personne.Rue ; Personne.Ville ;	Personne.Adresse	n :1	1 :1	1 :1
Personne.Age ; Personne.Nom ; Personne.Rue ; Personne.Ville ; Personne.Département	Personne.Age ; Personne.Prénom ; Personne.Nom Personne.NomJF Personne.Adresse ; Personne.DNum ; Département.DNum ; Département.DName	n :m	1 :n	1 :n

Toutes les cardinalités globales sont de 1 :1 sauf pour Schéma1.Personne.Nom qui participe à deux correspondances et possède donc une cardinalité globale de 1 :n.
Les cardinalités 1 :1 au niveau des instances signifient qu'une instance (ligne de table) de chaque schéma est utilisée pour matérialiser la correspondance. Les cardinalités 1 :n signifient que n instances (venant ici de tables différentes) du schéma 2 doivent être utilisées pour matérialiser la correspondance.

TAB. 3.2 – Exemples de cardinalités de matching présentes dans la figure 3.9

Classification des matchers

Dans [Shvaiko and Euzenat, 2005] les auteurs regroupent et complètent 3 études comparatives existantes concernant les algorithmes de recherche de correspondances, tout en établissant de nouvelles classifications (cf. figure 3.10) en se basant sur ces études. Ces classifications se basent sur 3 critères différents :

- Le traitement effectué sur les données afin d'obtenir les correspondances
- Le type des données prises en entrée
- Les propriétés générales de l'algorithme de matching

Dans leur première classification basée sur le traitement effectué sur les données fournies en entrée, les auteurs distinguent différentes catégories réparties selon une hiérarchie à deux niveaux.

Le premier niveau distingue le niveau élémentaire (qui analyse les entités du schéma prises isolément), du niveau structurel (qui tient compte de l'interaction entre les différents concepts présents dans les schémas).

Le deuxième niveau quant à lui distingue :

- *Les techniques syntaxiques* qui considèrent les concepts en se basant uniquement sur le texte et la structure sans se préoccuper de la sémantique cachée derrière la syntaxe. Ceci comprend notamment l'analyse des types de données des éléments, les mesures de similarité entre chaînes de caractères et l'analyse de représentations sous la forme de graphes.
- *Les techniques externes* qui font appel à des bases de connaissances externes tels des thesaurus afin d'obtenir davantage d'information sur la sémantique et les relations entre les concepts. Un utilisateur humain peut également être considéré comme ressource externe, si l'algorithme le sollicite afin de valider les actions à effectuer.
- *Les techniques sémantiques* qui se basent sur des modèles formels de représentation de la sémantique pour chercher les éléments correspondants des schémas fournis en entrée. Ceci comprend notamment l'utilisation d'un moteur d'inférence sur une représentation en Description Logic ou la vérification de la satisfaisabilité d'une représentation sous la forme de propositions logiques.

La deuxième classification se base sur le type de données prises en entrée par les algorithmes et distingue les catégories suivantes :

- Les algorithmes se basant sur des *entrées terminologiques* qui travaillent uniquement à l'aide de chaînes de caractères. Ceci comprend des techniques non linguistiques comme la similarité entre chaînes et des techniques linguistiques telles que l'utilisation de ressources linguistiques comme des thésauri, ou des techniques linguistiques telles que la tokenisation et la lemmatisation.
- Les algorithmes se basant sur des *entrées structurelles* c'est-à-dire sur les informations structurelles présentes dans les schémas au niveau interne en utilisant des informations propres à chaque élément comme leur type de données et au niveau externe en utilisant les relations entre éléments.
- Les algorithmes se basant sur des *entrées sémantiques* c'est-à-dire sur des modèles de représentation formelle des schémas comme la Description Logic, les formules propositionnelles ou des ontologies de haut niveau.

Si les deux premières catégories sont présentes directement dans le schéma, les entrées sémantiques nécessitent généralement un moteur d'inférence pour obtenir les correspondances.

La troisième classification contient les algorithmes utilisés dans les deux autres classifications et est elle-même encore subdivisée en sous-catégories :

- *Les techniques basées sur les chaînes de caractères* qui traitent les noms et descriptions des concepts en les considérant comme de simples chaînes de caractères. Ces techniques testent par exemple si un nom est préfixe/suffixe d'un autre ou tentent de découvrir les opérations nécessaires pour transformer le nom d'un concept en un autre (par ajouts, suppressions, substitutions de caractères).
- *Les techniques basées sur le langage* servent généralement à déblayer le terrain pour faciliter l'application d'autres techniques. Ces méthodes considèrent que les chaînes de caractères sont des phrases exprimées dans un langage naturel. Les mots (token) sont isolés, transformés en forme de base commune (ex : singulier) et sont ensuite filtrés afin d'éliminer les prépositions, les déterminants,etc.
- *Les techniques basées sur les contraintes* comparent les contraintes imposées aux éléments des schémas tels que le typage des données, l'intervalle de valeurs autorisées ou les cardinalités.
- *Les techniques utilisant des données linguistiques externes* tels les théraurus afin d'obtenir davantage d'information sur les relations existant entre deux termes comme par exemple la synonymie.
- *Les techniques de réutilisation de correspondances* qui utilisent les correspondances déjà trouvées entre o et o' et entre o' et o" afin d'améliorer et accélérer la recherche de correspondances entre o et o"
- *Les techniques basées sur les graphes*, où les schémas sont tout d'abord transformés en un graphe dont les noeuds représentent en général les concepts alors que les arcs représentent les relations entre concepts. Le principe des algorithmes se basant sur une telle structure est que la similarité peut se propager entre les noeuds via les relations. (ex : deux noeuds sont similaires si les noeuds adjacents sont similaires). La similarité peut également se baser sur les arcs (ex : si 2 relations provenant chacune d'un des schémas sont similaires et qu'une de leur extrémité est similaire, alors l'autre extrémité sera sûrement aussi similaire).
- *Les techniques basées sur les taxonomies* sont un cas particulier des techniques basées sur les graphes car elles ne considèrent que les relations hiérarchiques (IS-A).
- *Les techniques basées sur un modèle formel* transforment les schémas et correspondances possibles en propositions logiques dans un modèle formel (ex : description logic). La satisfaisabilité des correspondances est alors vérifiée.

Les différentes techniques présentées ci-dessus sont en général combinées afin de conserver les correspondances les plus fréquemment trouvées. Différents algorithmes peuvent par exemple être exécutés en parallèle, les résultats obtenus sont alors combinés par différentes techniques. Il se peut également que différents algorithmes soient exécutés séquentiellement. Dans ce cas, les techniques les plus rapides (en général moins précises) sont utilisées en premier afin d'éliminer les correspondances les moins pertinentes et d'alléger la tâche des algorithmes plus lourds. Il faut dans ce cas choisir et paramétrer correctement les algorithmes pour qu'aucune correspondance correcte ne soit éliminée au cours du processus.

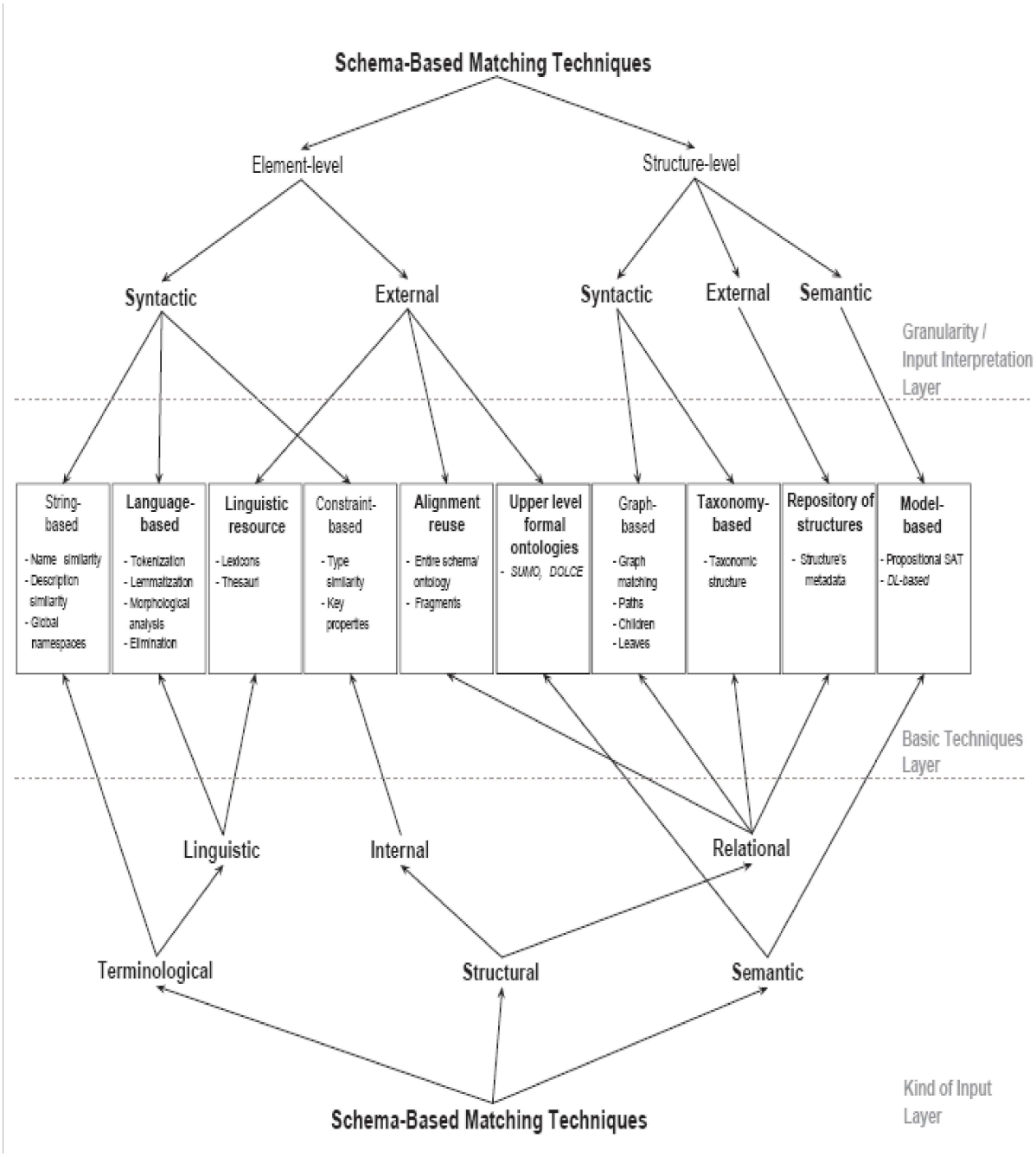


FIG. 3.10 – Classification des algorithmes de matching adoptée par [Shvaiko and Euzenat, 2005]

3.2 Intégration des instances

Nous avons maintenant parcouru de manière générale l'intégration de schémas. Une fois que l'on possède un schéma global produit à partir de divers schémas locaux, il faut également s'intéresser aux données. En effet, il est à noter que des données concernant le même objet du monde réel peuvent être contenues dans les différents schémas locaux. Il serait donc intéressant de pouvoir repérer et lier de telles instances afin de pouvoir fournir le maximum d'informations disponibles sur un objet lors de toute utilisation du schéma global. Cette opération est appelée *intégration des instances*.

Si chacun des différents schémas fournit un identifiant dont la valeur est identique pour les instances équivalentes, la découverte est relativement simple car il suffit de vérifier l'égalité entre les valeurs de cet identifiant. Cependant un tel identifiant est rarement présent. Il faut alors découvrir ce qui caractérise réellement chacune des instances (un ensemble de propriétés) afin de déterminer une base de comparaison et de décider de leur équivalence.

Les problèmes rencontrés lors de ce choix sont de deux types :

D'une part, les problèmes venant des schémas. Certaines propriétés ne sont présentes que dans une partie des schémas d'origine et ne peuvent donc être prises en compte pour attester de l'équivalence. Si on se base sur l'égalité des valeurs d'une telle propriété lors de la comparaison, les instances des différents sites ne peuvent être considérées comme identiques.

D'autre part, les problèmes venant des données. Comme nous l'avons vu précédemment, il est rare que les concepts présents dans les différents schémas se superposent parfaitement à cause de différents modèles de représentation ou de différentes vues du domaine selon le concepteur. Il en va de même pour les informations fournies par les différents sites. Ainsi plusieurs cas peuvent se présenter pour les données d'un même objet du monde réel :

- Les deux schémas fournissent des valeurs identiques pour une des propriétés de l'objet concerné.
- Les deux schémas fournissent des valeurs différentes pour une des propriétés de l'objet concerné.

Notons toutefois que la différence des valeurs d'une même propriété pour des individus identiques ne constitue pas nécessairement une erreur. En effet, certaines propriétés peuvent avoir des valeurs différentes selon le site, tout en étant valides dans leur contexte (ex : formats de mesures différents, avis de consommateurs, prix, etc.). Dans le cas d'une réelle erreur, celle-ci doit être corrigée, alors que dans le cas inverse, la valeur doit être conservée.

De plus, la valeur de certaines propriétés peut être identique, sans que les individus concernés soient identiques pour autant. Par exemple, le fait que deux articles soient vendus au même prix n'implique pas leur équivalence.

Ces différentes remarques nous conduisent à déduire que l'égalité entre valeurs de propriétés n'est pas appropriée lorsqu'aucun identifiant global[1] n'existe. Il vaut alors mieux utiliser des mesures de similarité entre valeurs de certaines propriétés et combiner ces mesures afin d'obtenir une similarité entre instances.

Ce type de problème a déjà été envisagé dans les bases de données sous le nom de *record linkage*. Dans [Gu et al., 2003], les auteurs distinguent la recherche par équivalence et par similarité sous les termes respectifs de *deterministic linkage* et de *probabilistic linkage*. Dans cet article, les

[1]Par identifiant global, nous entendons une propriété qui possède une valeur identique quel que soit le site où l'instance est présente. Ils ne faut pas les confondre avec les identifiants globaux au sens XML où la valeur littérale de l'attribut de type ID ne peut être affectée qu'à un seul élément du fichier XML.

auteurs présentent également un système typique de *record linkage* contenant les étapes essentielles pour mener à bien cette opération. Ces étapes, détaillées ci-dessous, sont la standardisation des données, le regroupement en blocs, la sélection des attributs, la comparaison et le choix du modèle de décision.

La standardisation des données

L'étape de standardisation consiste à ramener toutes les valeurs d'une même propriété à une structure identique. Sans cela, de nombreuses correspondances entre valeurs seraient manquées en raison de casses différentes, de diverses orthographes possibles pour un mot ou d'unités différentes. La méthode de standardisation doit être choisie en fonction du domaine auquel appartiennent les valeurs à standardiser car chaque domaine possède des particularités. Par exemple un nom anglo-saxon s'écrira *George W. Bush* alors qu'en français ce nom s'écrira *George Bush*. Dans ce cas, la standardisation pourrait consister à adopter une structure française pour tous les noms (l'ajout d'information inconnue étant impossible).

Le regroupement en blocs

Le regroupement en blocs consiste à réduire le nombre de comparaisons de paires de valeurs en regroupant à l'avance les paires d'individus susceptibles d'être identiques. Un bon attribut de regroupement doit posséder un large ensemble de valeurs distribuées de manière relativement homogène. Cet attribut doit en plus avoir une faible probabilité d'être erroné. Un code postal pourrait être un meilleur facteur de regroupement que le nom de famille car le taux d'erreurs d'encodage est plus faible.

La sélection des attributs

La sélection des attributs consiste à choisir les propriétés qui seront employées pour comparer les individus entre eux. Ces propriétés doivent être communes à tous les schémas d'origine. Il faut aussi que leurs valeurs contiennent suffisamment d'information pour permettre une comparaison de bonne qualité. Le sexe ne fournira par exemple que peu d'information car deux valeurs seulement sont possibles et de nombreuses personnes posséderont donc une même valeur pour cette propriété. Le nom de famille fournit davantage d'information (moins de personnes de même nom que de même sexe) mais possède un risque plus élevé d'être mal orthographié. Une solution à ce problème pourrait être de sélectionner tous les attributs communs à l'ensemble des schémas, les qualités de l'un compensant les défauts de l'autre.

La comparaison

L'étape de comparaison consiste à comparer les valeurs des attributs sélectionnés à l'étape précédente et à en évaluer la similarité. Pour ce faire, de nombreuses méthodes de similarité entre chaînes de caractères ont été développées (cf.[Cohen et al., 2003] pour une présentation et une comparaison de différentes mesures) et produisent une estimation de la similarité prenant sa valeur entre 0 et 1.

Le choix du modèle de décision

Le choix du modèle de décision consiste à choisir comment les similarités, calculées entre les valeurs des propriétés à l'étape précédente, seront combinées pour calculer une similarité entre individus. La méthode la plus simple consiste à calculer une moyenne des différentes similarités (éventuellement pondérée si certaines propriétés sont plus importantes que d'autres dans le calcul). Cette combinaison peut également se baser sur la fréquence d'apparition des valeurs similaires (ex : il est moins probable que deux individus dont le nom est "Dupont" soient identiques,

que deux individus de nom "Zabrinsky"). D'autres méthodes encore se basent sur les réseaux bayésiens ou sur d'autres modèles statistiques.

Exemple d'intégration des instances

En se référant à l'exemple présenté à la table 3.3, l'intégration des instances consisterait à déduire que Marcel Durant de T1 et M. Durant de T2 sont probablement la même personne car le nom, l'initiale du prénom ainsi que l'adresse sont les mêmes, toutefois sans certitude complète.

Si on examine Dupont J. de T2, on s'aperçoit que 2 instances de T1 pourraient lui correspondre alors que l'adresse est inconnue dans T2. On pourrait aussi se baser sur le numéro de référence qui est identique pour Dupont J. et Dupont Jules. Celui-ci cependant n'est pas identique pour Durant M. et Durant Marcel.

Plusieurs conclusions sont alors possibles :

1. La référence n'est pas standard dans les deux tables. On ne peut donc se baser sur celle-ci pour distinguer les instances identiques, auquel cas Dupont J. peut :
   – correspondre à Dupont Jules
   – correspondre à Dupont Julien
   – être quelqu'un d'autre

   De même, Durant M. peut être :
   – Durant Marcel
   – un autre habitant de la même maison et de même initiale de prénom
2. La référence est un standard pour les deux tables, auquel cas Dupont J. est Dupont Jules alors que Durant M. n'est pas Durant Marcel

C'est donc à l'expert du domaine de prendre la décision finale quant à l'équivalence ou non des différentes instances.

Ref	Nom	Prénom	Adresse
CM03XZ	Dupont	Jules	rue de la Paix, 7 1985 Wousiplou
RT543F	Dupont	Julien	rue du 11 Novembre, 56 1945 Les Bains
CRX321	Durant	Marcel	avenue du Pont,4 3452 Machin-Sur-Seine

T1

Ref	Nom	Prénom	DNaiss	Adresse
CM03XZ	Dupont	J.	21/04/1954	—
CZE384	Durant	M.	25/11/1972	avenue du Pont,4 3452 Machin-Sur-Seine

T2

Tab. 3.3 – Exemple d'intégration des instances

3.3 Etat de l'art

3.3.1 Intégration de schémas/ontologies et recherche des correspondances

Comme cela a déjà été mis en évidence précédemment, de nombreuses techniques et outils ont été développés pour retrouver les correspondances entre différents schémas ou ontologies et afin de les intégrer. Il s'avère donc difficile de présenter un état de l'art exhaustif et reflétant exactement l'état actuel de la recherche dans ce domaine. Le présent état de l'art se basera donc principalement sur un article de Y. Kalfoglou et M. Schorlemmer [Kalfoglou and Schorlemmer, 2005] effectuant un relevé des principales méthodes, théories et outils se rapportant à l'intégration d'ontologies. Ceci sera complété par des références à des articles décrivant une méthode ou un outil spécifique non mentionné dans [Kalfoglou and Schorlemmer, 2005].

Dans [Fernández-Breis and Martínez-Béjar, 2002], les auteurs décrivent une méthode permettant le développement d'ontologies globales dans un environnement distribué. Chaque utilisateur du système peut se trouver dans un endroit différent du globe et travailler sur l'ontologie à n'importe quel moment de la journée. Dans le système développé, deux types d'utilisateurs sont distingués : les utilisateurs dits "normaux" et les experts.

Les utilisateurs normaux peuvent simplement consulter l'ontologie globale et exécuter des requêtes. Les experts quant à eux peuvent influer sur le contenu de l'ontologie, en soumettent au système leur propre ontologie concernant le domaine. Afin d'assurer le bon fonctionnement du système, l'expert doit spécifier pour chaque concept de son ontologie, ses parents et ses descendants dans la taxonomie, ses attributs, ses relations et ses synonymes. La spécification des synonymes permet à chaque utilisateur de pouvoir interroger l'ontologie globale selon son propre vocabulaire.

Le système possède également une protection des ontologies individuelles contre la modification par un autre utilisateur que l'auteur et empêche l'accès aux parties d'ontologies non validées par leur auteur. Ceci permet de garantir la cohérence de l'ontologie globale. Le système aidera les experts à produire une ontologie globale en analysant les différentes ontologies individuelles.

Les auteurs introduisent aussi la notion de *gain de connaissance* afin de mesurer l'efficacité de leur outil. Ce gain de connaissance est considéré pour chaque expert et représente le nombre de concepts relations ou attributs qui n'étaient pas présents dans l'ontologie de cet expert et qui le sont dans l'ontologie globale.

Dans [Maedche et al., 2002], les auteurs défendent une thèse selon laquelle la production d'une ontologie globale virtuelle est la méthode la plus efficace pour tenir compte de la nature distribuée du Web. En effet, la simple définition de correspondances entre les ontologies, matérialisées par des ponts sémantiques (*semantic bridges*), permet aux utilisateurs de continuer à utiliser leurs ontologies propres tout en participant à une ontologie globale donnant accès aux connaissances d'autres experts.

Le système développé par les auteurs (MAFRA) se découpe selon 2 dimensions : la dimension horizontale et la dimension verticale. Les modules horizontaux représentent les différentes étapes du processus d'intégration alors que les modules verticaux interviennent tout au long de ce processus.

Le processus utilisé dans MAFRA se présente comme suit : les ontologies sont tout d'abord ramenées dans une syntaxe commune (RDFS) afin de mieux mettre en évidence les différences sémantiques. Les correspondances entre les ontologies sont ensuite recherchées en combinant diverses techniques telles que la similarité lexicale ou la similarité via les propriétés associées à chaque concept. La similarité est propagée de manière top-down mais aussi bottom-up dans la hiérarchie (si 2 concepts sont similaires, leurs parents et descendants le sont sans doute aussi). Les correspondances découvertes lors de l'étape précécente sont ensuite matérialisées sous la forme de ponts sémantiques entre les ontologies pour permettre la traduction des instances de chaque

concept en instances du concept correspondant dans une autre ontologie. Un traducteur implémentant les *ponts sémantiques* est ensuite produit. Une étape finale de *post-processing* contrôle la traduction et tente d'améliorer celle-ci. Afin de représenter les *ponts sémantiques*, les auteurs ont défini une ontologie spécifique nommée SBO (Semantic Bridging Ontology).

Dans [Calvanese et al., 2001], les auteurs défendent une thèse semblable aux précédentes. En effet, selon eux les ontologies locales ne doivent pas être vues comme une étape intermédiaire permettant d'arriver à une ontologie globale. Selon eux l'ontologie globale ne doit pas contenir de données en tant que telles, mais doit accéder aux ontologies locales afin de répondre aux requêtes des utilisateurs. Les auteurs développent le fait qu'un concept d'une ontologie peut en général être spécifié comme une vue sur les autres ontologies, les correspondances pouvant donc être représentées par des requêtes, à condition qu'un langage de requêtes suffisamment puissant soit associé au langage des ontologies.

Les auteurs définissent la notion d'*Ontology Integration System (OIS)* comme un triplet $<\mathcal{G},\mathcal{S},\mathcal{M}_{\mathcal{G},\mathcal{S}}>$. G étant l'ontologie globale, S l'ensemble d'ontologies locales et M l'ensemble des correspondances entre G et S. Une définition formelle des approches LaV, GaV et mixte est également proposée ainsi qu'une formalisation spécifique pour des ontologies exprimées à l'aide de la *logique des descriptions* (description logic).

Dans [Madhavan et al., 2002], les auteurs définissent un système permettant d'exprimer des correspondances entre schémas et ontologies quel que soit leur format et ce sans passer par un modèle intermédiaire commun. Toutefois, un tel modèle est prévu afin de pouvoir représenter les correspondances même si leur expression directe entre les deux modèles de départ est impossible ou trop complexe.

Le langage utilisé pour représenter ces correspondances varie en fonction des modèles correspondant aux schémas/ontologies à comparer. Les auteurs mettent également en évidence le fait qu'il n'existe pas qu'un seul ensemble de correspondances correct. En effet, la validité d'une correspondance peut dépendre de la tâche pour laquelle elle est considérée. Il faudrait donc être en mesure de savoir si un ensemble de correspondances est correct pour une tâche particulière. Les auteurs décrivent également les propriétés que devrait posséder une correspondance idéale.

Dans [Giunchiglia et al., 2005a] [Giunchiglia et al., 2005b] les auteurs proposent une approche originale de recherche de correspondances, le matching sémantique. Cette méthode consiste à produire les mappings entre deux ontologies en se basant non pas sur la similarité entre les labels des concepts des ontologies mais sur les relations sémantiques entre ceux-ci.

Les auteurs se basent sur une représentation des ontologies sous la forme d'arbres et s'appuient sur deux notions principales pour effectuer la recherche de correspondances entre les concepts, les *concepts des labels* et les *concepts des noeuds*.

Le *concept d'un label* représente l'ensemble des documents qui parlent de la signification du label. Ce type de concept est indépendant de la localisation du label dans l'arbre et dépend donc seulement du label lui-même.

Le *concept d'un noeud* représente l'ensemble des documents qui seront réellement classés sous ce noeud de l'arbre. Il consiste donc en l'intersection des concepts de labels des noeuds ancêtres et du noeud lui-même. Ce type de concepts dépend donc de la position du noeud dans l'arbre.

Si l'on reprend l'exemple utilisé par les auteurs dans [Giunchiglia et al., 2005a] et présenté à la figure 3.11, quel que soit l'arbre et la position dans celui-ci, le contexte du label "Europe" représentera l'ensemble des documents à propos de l'Europe. Dans le schéma A1, le concept du noeud "Europe" sera l'ensemble de documents étant des *images* à propos de l'*Europe*. Dans le cas du schéma A2 les 2 types de concepts sont identiques (le noeud étant la racine de l'arbre et

ne possèdant donc pas d'ancêtres).

Les auteurs représentent les concepts à l'aide d'un langage logique constitué d'éléments atomiques qui peuvent être composés à l'aide de connecteurs ensemblistes pour obtenir des formules plus complexes. Les relations sémantiques entre les concepts de noeuds de chaque paire de noeuds (venant chacun d'une ontologie différente) sont converties en relations propositionnelles dont la satisfaisabilité est vérifiée. Le problème de recherche de correspondances est donc transformé en un problème de validité propositionnelle.

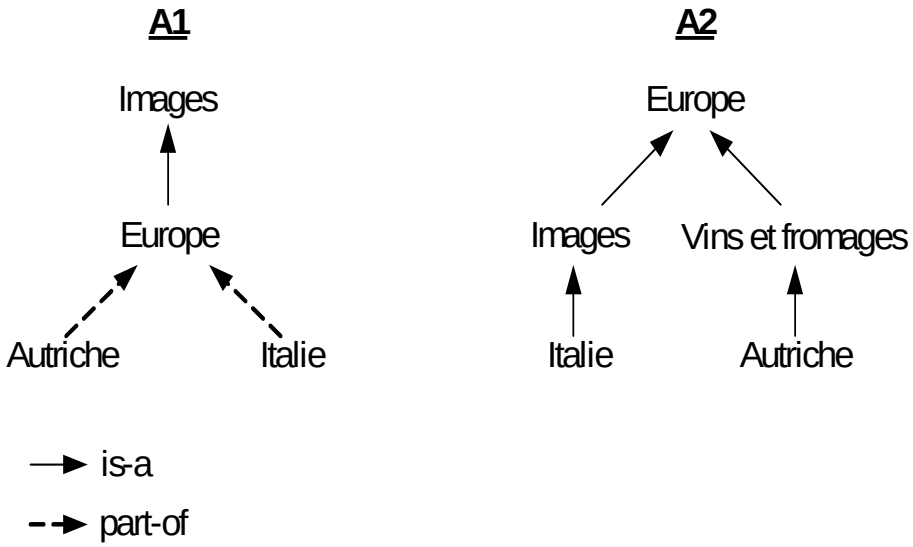


FIG. 3.11 – Exemple de matching avec S-Match

Prompt [Noy and Musen, 2003] est une suite d'outils de gestion de plusieurs ontologies développée comme un plugin à l'outil de développement d'ontologies Protégé [2]. Prompt permet d'effectuer diverses tâches sur les ontologies comme la fusion de deux ontologies, la comparaison de deux versions d'une même ontologie, l'extraction d'une partie d'une ontologie ou le déplacement de parties d'une ontologie vers une autre.

Lors d'une fusion de deux ontologies, Prompt suggère les mappings qui lui semblent les plus pertinents afin de fusionner les concepts dans l'ontologie finale. Pour effectuer ses choix, il se base sur la structure de l'ontologie, sur les relations entre les concepts et également sur les informations fournies par l'utilisateur. L'outil est également à même de repérer les conflits qui pourraient surgir suite à l'exécution d'une opération et propose automatiquement des solutions.

Dans [Stumme and Maedche, 2001], les auteurs introduisent une méthode d'intégration d'ontologies par fusion. L'outil utilise tout d'abord des techniques de traitement de la langue naturelle pour extraire les ontologies et instances d'un ensemble de documents. Les ontologies produites sont ensuite intégrées semi-automatiquement avec l'aide d'un expert maîtrisant la connaissance de ce domaine. Afin que la méthode puisse fonctionner, il est indispensable que les documents soient représentatifs du domaine concerné. Ils doivent aussi couvrir tous les concepts des ontologies et les séparer suffisamment. Le processus d'intégration comprend un mécanisme de "population" des ontologies par des instances extraites des documents fournis en entrée. Ceci permet d'avoir les instances nécessaires au processus d'intégration, car peu d'ontologies possèdent d'origine des instances en nombre suffisant.

Dans [Doan et al., 2002], les auteurs présentent une méthode de matching se basant sur le

[2] `http://protege.stanford.edu`

machine learning. La décision de la similarité se fait à l'aide de la *joint probability distribution* qui consiste à calculer les probabilités suivantes P(A,B) , P($\overline{A}$,B), P(A,$\overline{B}$), P($\overline{A}$,$\overline{B}$) (P(A,B) signifiant *probabilité d'appartenir à l'ensemble A et à l'ensemble B*). Ces probabilités servent à calculer des mesures de similarité. Dans cette approche, ces probabilités sont calculées par combinaison des résultats de deux classifieurs (un classifieur décidant de l'appartenance à l'ensemble A et un autre décidant de l'appartenance à l'ensemble B). Ces classifieurs se basent sur différents critères, tant de la structure que de la hiérarchie ou des instances (ex : valeur des instances, nom des instances,...). Les prédictions des "learners" implémentant les différents critères sont ensuite combinées à l'aide d'un "meta-learner" afin de décider si une instance appartient à l'ensemble (A $\cap$ B).

Dans [Mitra et al., 2000], les auteurs mettent en évidence les points faibles d'une approche d'intégration consistant à produire une grosse ontologie contenant l'ensemble des données des ontologies locales. En effet, selon eux, ce type d'approche n'est pas du tout adapté aux ontologies de grande taille ni lorsque les ontologies ou données locales changent régulièrement. Il faudrait donc préférer une approche établissant des liens sémantiques (exprimés sous la forme de règles d'articulation) entre les différentes ontologies et se baser sur ceux-ci pour rassembler l'information de plusieurs ontologies. Les ponts sémantiques entre ontologies sont eux-mêmes représentés par une ontologie dite *d'articulation*. L'article décrit également un outil (ONION) permettant de découvrir les correspondances entre plusieurs ontologies et de définir une ontologie contenant les ponts sémantiques. Cet outil représente les ontologies sous la forme de graphes. Une algèbre permettant d'exprimer les ponts sémantiques et d'exécuter des requêtes sur les ontologies est également présentée.

Etant donné le nombre de travaux existants sur le thème de l'intégration d'ontologies et de schémas, il est impossible d'en donner ici une liste exhaustive. Le lecteur souhaitant approfondir la problématique pourra trouver davantage d'explications et de références notamment dans [Rahm and Bernstein, 2001],[Kalfoglou and Schorlemmer, 2005], et [Shvaiko and Euzenat, 2005].

3.3.2 Intégration des instances

L'intégration des instances fait, comme l'intégration de schémas, l'objet de nombreux travaux de recherche dont certains sont présentés ci-dessous.

Dans [Guha, 2004], l'auteur propose tout d'abord une méthode simple qui consisterait à affecter à toute instance une URI et démontre l'impossibilité de l'utiliser car il faudrait l'imposer lors de tout usage de cette instance. De plus même si l'ensemble de la planète convenait d'une URI pour une instance, le problème ne serait pas résolu car de nouvelles instances sont créées tous les jours. Ensuite, une approche basée sur des clés (identifiantes) est proposée. Pour cela deux types de clés sont distingués : les "common keys", les "shared keys".

Soit A et B deux bases de données ou ontologies :

- Une common key k entre A et B signifie que k est une clé identifiante dans A et dans B mais que si une même instance i est présente dans A et dans B, la valeur de k ne sera pas nécessairement identique.
- Une shared key k entre A et B signifie que k est une clé identifiante dans A et dans B et que si une même instance i est présente dans A et dans B, la valeur de k sera d'office identique. k est donc une clé identifiante sur A $\cup$ B.

L'auteur met également en évidence le problème des clés contenant de nombreux attributs dont la valeur est rarement égale pour 2 instances identiques et introduit la notion de *Discriminant description* (DD) : ce sont des formules qui ne sont satisfaites que par une instance et qui la distinguent donc des autres.

L'approche d'intégration proposée par l'auteur est une approche probabiliste : elle consiste à trouver une DD commune à A et B afin que la probabilité que deux instances identiques (re-

présentant le même objet) satisfassent cette DD soit proche de 1. L'auteur propose également la meilleure manière de calculer la probabilité qu'une instance satisfasse une DD.

Dans [Sarawagi and Bhamidipaty, 2002], les auteurs proposent une approche basée sur le *Machine learning*. Celle-ci consiste à fournir comme entraînement à un *learner* un ensemble d'instances identiques et un autre ensemble d'instances différentes mais susceptibles d'être considérées comme identiques (forte similarité). Le *learner* est ensuite capable de distinguer les paires identiques des non-identiques dans un grand ensemble de données.

La création d'ensembles d'entraînement est relativement lourde car il s'agit généralement d'un processus manuel. Les auteurs proposent donc une approche originale permettant de produire automatiquement ces ensembles. Pour ce faire, ils se basent sur des mesures de similarité différentes et exploitent les différences entre les résultats de celles-ci afin de produire les ensembles.

Cette approche se distingue également par le fait que les ensembles d'entraînement ne sont pas statiques mais sont choisis dynamiquement parmi l'ensemble des instances existantes. L'utilisateur valide et corrige les ensembles produits qui sont ensuite injectés dans le *learner*. Si l'utilisateur n'est pas satisfait des résultats produits par le *learner*, il peut demander la génération de nouveaux ensembles d'entraînement.

Dans [Doan et al., 2003], les auteurs décrivent une approche qui pourrait plutôt se définir comme complément à d'autres méthodes d'intégration des instances.

Cette méthode se base sur des *profilers* qui sont des outils possédant une connaissance approfondie sur un concept particulier d'un domaine (ex : un enfant a un âge < 12 ans). Un tel outil peut donc décider si une instance correspond bien aux différentes contraintes imposées pour appartenir à une certaines classe (ex : une personne de 56 ans n'est pas un enfant).

L'approche développée ici se base sur des informations non encore utilisées par les autres approches d'intégration des instances, à savoir les attributs n'ayant obtenu aucune correspondance suite au matching des schémas et qui constituent la spécificité des instances de chacun des schémas. La méthode consiste tout d'abord à appliquer n'importe quelle technique d'intégration des instances afin de ne retenir que les paires d'instances suffisamment similaires pour être susceptibles d'être équivalentes. Ensuite, l'instance constituée par l'union des attributs des instances de chaque paire est soumise à une série de profilers qui permettront de décider si les valeurs de différents attributs est bien cohérente par rapport à la définition générique que l'on possède de la classe de la paire.

Supposons que nous possédions deux schémas fournissant des informations sur des personnes : l'un d'eux fournit une date de naissance et l'autre une date de décès. Si deux instances sont considérées comme similaires par une autre méthode d'intégration des instances, mais que la date de décès est antérieure à la date de naissance, un profiler spécifique permettra de détecter cette incohérence.

L'article [Lin, 1998] ne porte pas à proprement parler sur l'intégration des instances mais peut lui servir de support. L'auteur propose une définition de la similarité et la façon de la mesurer. La définition proposée est pensée de manière à être universelle et est applicable tant que le domaine d'emploi peut être exprimé à l'aide d'un modèle probabiliste. La correction de cette définition se base sur un ensemble d'hypothèses dont elle découle directement. Donc si on est d'accord avec l'ensemble d'hypothèses, on sera d'accord avec la définition. Cette mesure est ensuite appliquée à différents domaines : la similarité de valeurs ordinales, de chaînes de caractères, de mots et de taxonomies.

Deuxième partie

Méthodologie et outils

Chapitre 4

Méthodologie

Dans ce chapitre, nous commencerons par justifier les choix des ontologies pour représenter la structure et les données des sites web. Ensuite, nous présenterons rapidement les différentes étapes du processus d'intégration de données issues de sites web et ainsiqu'une vue schématique de l'ensemble de ce processus. Enfin nous détaillerons une par une chacune des étapes précitées. Il est à noter que la partie d'intégration de la structure a été élaborée par amélioration d'une approche existante, alors que l'intégration des instances a été développée lors de mon stage à l'EPFL[1] durant le premier semestre de l'année académique 2005-2006.

4.1 Choix du format de représentation des structures des sites web

Dans ce mémoire, les ontologies ont été choisies pour représenter la structure des sites web à intégrer. Plusieurs raisons sont à l'origine de ce choix.

Tout d'abord, comme mentionné à la section 2.1, une ontologie est une représentation des concepts d'un domaine et de leurs relations. Si une telle ontologie existe déjà pour le domaine couvert par les sites web à intégrer, elle constitue une bonne base pour le processus d'intégration, à plus forte raison si cette représentation est communément acceptée par les experts du domaine. Cette ontologie peut si nécessaire être enrichie au fur et à mesure de l'intégration de nouveaux sites.

Ensuite, l'approche adoptée par le Semantic Web est l'utilisation d'ontologies pour annoter et structurer les sites web et permettre une compréhension plus aisée du contenu des sites web par des machines. Le processus d'intégration développé dans ce mémoire étant l'intégration de sites web, les ontologies s'imposaient donc d'emblée.

4.2 Vue schématique de la méthode

La figure 4.1 présente une vue schématique du processus d'intégration de sites web présenté ci-dessous.

La première étape consiste à choisir les différents sites que l'on désire intégrer, à choisir l'ensemble de pages représentant le concept dont on veut intégrer les informations (ex : un projet) et à en extraire la structure et les données sous la forme de fichiers OWL à l'aide de Retrozilla (cf. section 5.1) (*Ontologie de la structure du site i* et *Ontologie des données du site i* sur la figure 4.1).

[1] Ecole Polytechnique Fédérale de Lausanne

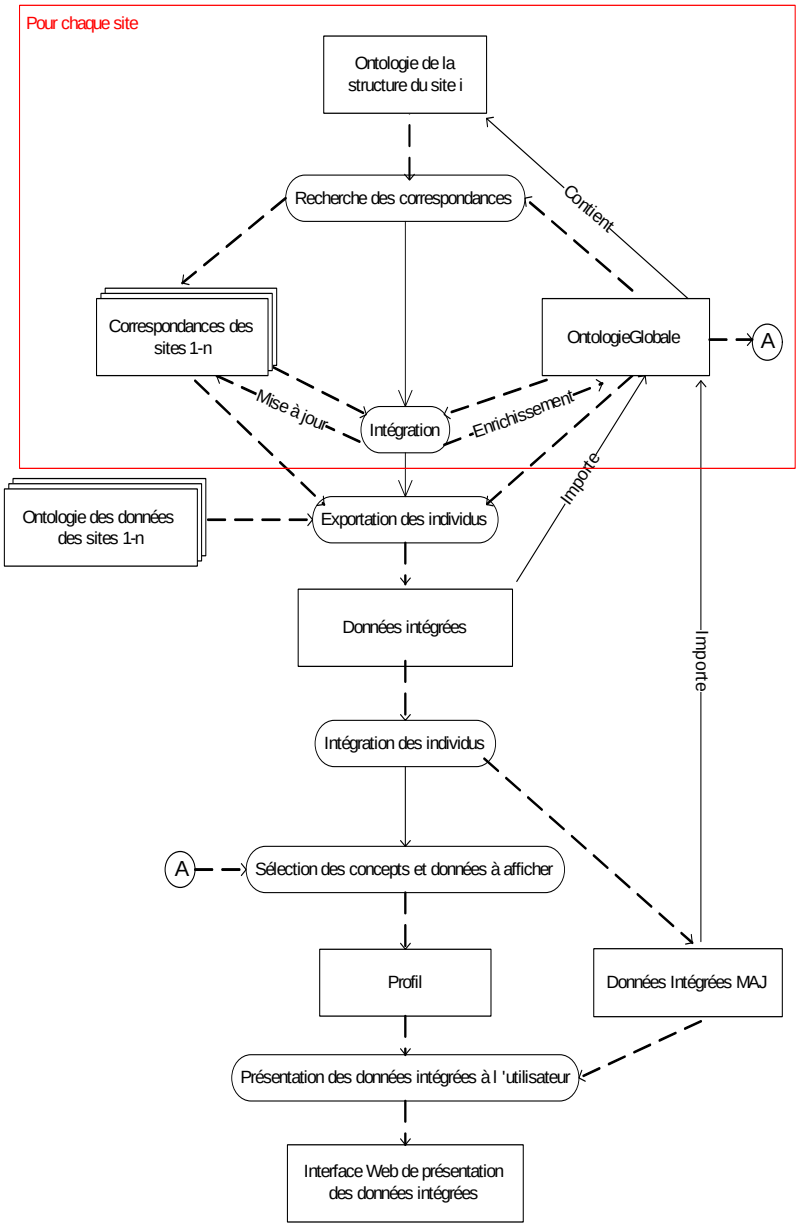


FIG. 4.1 – Schéma général de la méthode d'intégration des sites web adoptée dans ce mémoire.

Une fois que nous disposons d'une représentation de la structure des différents sites sous la forme d'une ontologie OWL, nous pouvons passer à l'étape suivante. Les ontologies sont tout d'abord classées par ordre de fiabilité. Ensuite, l'ontologie du site web jugé le plus fiable est considérée comme étant la première *Ontologie Globale*(cf. figure 4.1). Cette dernière est ensuite comparée avec l'ontologie du second site (*Structure Site i* sur la figure 4.1) afin de trouver les correspondances (*Correspondances des sites 1-n* sur la figure 4.1) existant entre elles. Ces correspondances sont alors utilisées pour enrichir l'ontologie globale qui sera par la suite comparée et intégrée avec l'ontologie contenant la structure du prochain site web. Les correspondances entre les ontologies des différents sites web déjà intégrés et l'ontologie globale sont mises à jour pour tenir compte des modifications apportées à l'ontologie globale. Ce processus se poursuit de manière incrémentale jusqu'à ce que plus aucun site ne doive être intégré. Il est à noter que si, dès le début du processus d'intégration, une ontologie du domaine bien acceptée existe, celle-ci peut être considérée comme l'ontologie d'un site web supplémentaire qui serait plus fiable que les autres.

Maintenant que nous possédons la structure de l'ontologie globale, il nous faut également nous intéresser aux instances. En effet, les individus des ontologies d'origine doivent être exportés dans un format adapté à l'ontologie globale (cf. section 4.3.3). Afin de réaliser cela, il est nécessaire d'utiliser la liste des correspondances entre les éléments de l'ontologie du site web et ceux de l'ontologie globale (*Correspondances Site i* sur la figure 4.1) afin de savoir comment chaque classe et propriété doit être traduite.

Nous disposons à présent d'une ontologie intégrant la structure des différents sites, ainsi que d'une ontologie contenant les données des divers sites. Il nous reste encore à constater que les individus présentés par un site pourraient également figurer sur un autre site. Chaque site fournissant en général des informations différentes dans leur contenu, il serait intéressant de repérer ces individus représentant le même objet du monde réel afin de pouvoir combiner leurs informations pour en présenter une vision unique à un utilisateur (cf. section 4.3.4).

La dernière étape du processus d'intégration consiste à produire une version aisément accessible pour tout un chacun du contenu de l'ontologie globale. L'origine des données étant des sites web, il semblait judicieux d'utiliser également l'Internet comme média d'affichage du contenu de cette ontologie (cf. section 4.3.5).

4.3 Processus d'intégration

4.3.1 Recherche des correspondances

La méthode développée dans cette section est issue de celle développée dans les articles suivants [Noy and Musen, 2000, Noy and Musen, 2001, Noy and Musen, 2003]. Cette méthode automatise autant que possible le processus de recherches de correspondances et facilite l'exécution de ce processus par un expert grâce à une grande interactivité.

Dans cette méthode, les ontologies sont considérées comme des graphes dont les noeuds sont les classes et les arcs sont les propriétés. Un arc existe entre une classe A et une classe B s'il existe une propriété possédant A dans son domaine et B dans sa cible.

Les relations d'héritage sont considérées différemment, et ne sont pas représentées en tant qu'arcs sur le graphe. En effet, chaque classe ainsi que ses sous-classes et classes parentes sont considérées comme constituant un noeud unique appelé *groupe d'équivalence*. Les arcs entrant dans ce noeud représentent l'ensemble des propriétés possédant au moins une des classes du groupe dans leur cible. Les arcs sortants quant à eux représentent l'ensemble des propriétés possédant au moins une des classes du groupe dans leur domaine.

Les correspondances repérées par cette méthode sont des couples d'éléments dont chaque membre provient d'une ontologie différente.

La méthode possède deux algorithmes qui seront détaillés par la suite. L'un se base sur des données purement locales pour juger de la similarité. Pour des classes par exemple, il ne se base que sur les propriétés directement attachées à la classe ainsi qu'aux classes cibles de ces propriétés. L'autre algorithme parcourt les chemins du graphe entre deux classes possédant une correspondante dans l'autre ontologie afin de repérer sur ces chemins d'autres classes et/ou propriétés équivalentes.

Concernant l'algorithme "local", une série d'opérations possibles ont été définies à savoir la fusion de deux classes, la fusion de deux propriétés, la fusion d'instances, la copie d'une classe, la copie "profonde" d'une classe (on copie non seulement la classe mais aussi tout ce qui lui est rattaché via des propriétés). Pour chaque opération, les changements à effectuer ainsi que les conflits possibles ont été prévus (cf. exemple de la fusion de classe ci-dessous). Au fur et à mesure des opérations validées par l'expert, une troisième ontologie contenant les résultats des opérations est construite. Celle-ci représente la fusion des deux autres.

Par exemple, l'opération de fusion de deux classes A et B est définie comme suit :
- Créer une classe M dont on choisit le nom dans l'ontologie globale. Cette classe représente la fusion des deux classes d'origine.
- Pour chaque super-classe ou sous-classe C_i des classes A et B, la copier dans l'ontologie fusionnée et définir leurs copies respectivement comme des super-classes et sous-classes de la classe M.
- Pour chaque propriété attachée aux classes A et B, les copier dans l'ontologie globale et les attacher à la classe M. Si des restrictions de cardinalités étaient associées à une des classes A et/ou B, copier ces restrictions dans la classe M. Si une propriété était attachée simultanément aux classes A et B, les restrictions de cardinalités associées doivent être copiées dans la classe M en gardant le plus grand intervalle de cardinalités (cf. table 4.1).
- Pour chaque paire de propriétés similaires lexicalement, suggérer de fusionner ces deux propriétés (en tenant compte du plus grand intervalle de cardinalités).
- Pour chaque paire de super-classes ou sous-classes (l'une de la classe A et l'autre de la classe B), similaires lexicalement, suggérer de fusionner ces classes. Leur similarité est d'autant plus forte qu'elles sont super-classes ou sous-classes de deux classes que l'expert a considérées comme équivalentes (car il en a demandé la fusion dans l'ontologie globale).

Les conflits possibles dans ce processus sont :
- Une ou plusieurs classes appartenant à la cible d'une propriété de l'ontologie fusionnée ne sont pas encore présentes dans l'ontologie globale. L'algorithme suggère alors de copier ces classes dans l'ontologie globale ou de supprimer la propriété de l'ontologie globale.
- Une ou plusieurs classes appartenant au domaine d'une propriété de l'ontologie fusionnée ne sont pas encore présentes dans l'ontologie globale. L'algorithme suggère alors de copier ces classes ou de supprimer la propriété de l'ontologie globale.
- Un élément de même nom qu'un élément que l'on veut copier est déjà présent dans l'ontologie globale. L'algorithme propose alors de fusionner les deux éléments ou de renommer l'un d'eux.

Lors de son exécution, l'algorithme local commence par proposer une liste d'opérations de fusion et de copie des classes des deux ontologies comparées selon que les classes possèdent ou non une classe similaire lexicalement dans l'autre ontologie. Ensuite, au fur et à mesure que l'expert valide les opérations de la liste ou en crée de nouvelles, l'algorithme met à jour la liste des opérations suggérées, comme spécifié dans la définition des différents types d'opérations (cf. l'exemple sur la fusion de classes ci-dessus).

L'algorithme "global" quant à lui se base sur les hypothèses suivantes :
- Si deux paires de termes sont similaires et qu'il existe des chemins reliant les éléments de chaque paire entre eux, alors il y a de fortes chances pour que les éléments intermédiaires présents sur les chemins soient eux aussi similaires.
- Les créateurs de différentes ontologies relient des concepts similaires de la même manière, même si les concepts ne portent pas le même nom.

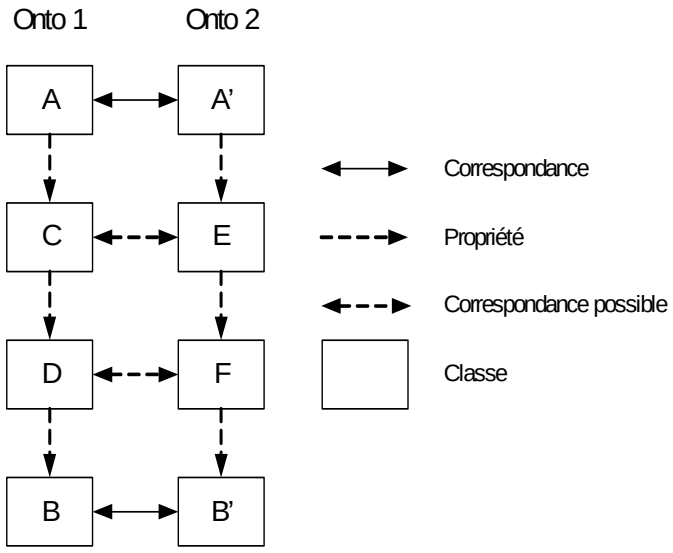


FIG. 4.2 – Exemple du fonctionnement de l'algorithme global de recherche des correspondances.

Cet algorithme prend donc en entrée deux paires d'éléments reliés par une relation de correspondance ([A,A'] et [B,B'] dans la figure 4.2). Celles-ci peuvent avoir été découvertes par l'algorithme "local" ou lors d'une précédente exécution du présent algorithme voire même avoir été définies par un expert. L'algorithme définit ensuite tous les chemins de longueur inférieure à une certaine valeur entre les éléments des deux paires provenant de la même ontologie. Il rassemble alors les chemins de même taille, parcourt ceux-ci deux à deux (chacun provenant d'une ontologie différente) ([A,C,D,B] et [A',E,F,B'] dans la figure 4.2) et augmente d'une constante X la mesure de similarité entre les éléments se situant en même position sur ces chemins([C,E] et [D,F] dans la figure 4.2).

La constante X varie si l'on a affaire à une simple classe ou à un groupe d'équivalence. En effet, si l'algorithme repère une équivalence entre une classe W et un groupe d'équivalence [Y,Z], la similarité entre W et Y ne peut être considérée comme aussi forte que si l'algorithme avait trouvé simplement Y sur le chemin à la place du groupe [Y,Z]. La similarité finale entre deux noeuds est produite par addition des mesures obtenues lors du parcours de chacune des paires de chemins. Cette mesure de similarité reflète donc le nombre de fois que les deux classes apparaissent en même position dans un chemin et évite que des classes totalement distinctes mais apparaissant en même position dans un seul des chemins ne soient considérées comme similaires.

Ce second algorithme possède cependant certaines faiblesses. Son efficacité dépend effectivement des ontologies traitées. Si l'on considère une première ontologie présentant peu de propriétés et beaucoup de classes organisées selon une grande hiérarchie, et une seconde ontologie possédant davantage de propriétés mais une hiérarchie réduite, beaucoup de noeuds de la première ontologie sont regroupés en un seul dans la seconde et de ce fait il est peu aisé de trouver des chemins de longueur équivalente entre les paires d'éléments.

Dans son ensemble, cette méthode de recherche des correspondances possède une faiblesse. En effet, elle ne permet de repérer que des correspondances simples c'est-à-dire reliant un élément d'une des ontologies à un seul élément de l'autre ontologie, alors que dans certains cas des correspondances impliquant plusieurs éléments d'au moins une des ontologie devraient pouvoir être repérées (ex : o1.Nom = concat(o2.Prénom,o2.Nom)). Cette faiblesse peut cependant être contournée si on s'assure, lors de la phase d'extraction, de toujours définir les propriétés des concepts avec la granularité la plus fine possible (ex : ne pas extraire la propriété PrénomNom

mais plutôt lui préférer deux propriétés distinctes Prénom et Nom). De plus, si un concept n'est présent que sous la forme d'une chaîne de caractères dans un des sites mais est susceptible de posséder davantage de propriétés dans un autre site, il est préférable de définir ce concept comme une classe (ex : un site fournit seulement le nom du responsable d'un projet, alors qu'un autre site pourrait fournir davantage d'information quant à cette personne comme par exemple sa date de naissance).

4.3.2 Intégration de la structure

Lors de l'intégration de la structure, les actions à effectuer pour intégrer les différents composants des ontologies (classes, propriétés et restrictions de cardinalités) doivent être envisagées et sont présentées ci-dessous.

Classes

Lors de l'intégration des classes, plusieurs cas peuvent se présenter :

D'une part, tout couple de classes des ontologies sources considérées comme se correspondant lors de l'étape précédente formera une seule classe dans l'ontologie globale. Les propriétés, sous-classes et super-classes de chacune de ces classes doivent être copiées dans l'ontologie globale et associées à la nouvelle classe. Ensuite, les propriétés possédant pour cible l'une des classes d'origine, doivent être mises à jour dans la nouvelle ontologie afin que leur cible contienne la nouvelle classe. Enfin, les propriétés de la nouvelle classe considérées comme similaires doivent être fusionnées dans l'ontologie globale.

D'autre part, les classes ne possèdant aucun correspondant dans l'autre ontologie peuvent être copiées directement dans l'ontologie globale ainsi que les propriétés qui leur sont attachées et leurs super-classes et sous-classes (et leurs propriétés).

Si pour une raison ou une autre, une classe n'est pas jugée importante pour le but d'utilisation de l'ontologie globale, elle peut être simplement laissée de côté et n'être ni copiée ni fusionnée avec une autre. Les propriétés dont elle est l'unique cible ou l'unique domaine ne doivent alors pas non plus apparaître dans l'ontologie globale.

Propriétés

Lorsque deux propriétés sont considérées comme se correspondant, une seule propriété sera présente dans l'ontologie globale. Les domaines et cibles de cette propriété seront l'union respectivement des domaines et cibles des propriétés d'origine. Si des correspondances avaient été repérées lors de l'étape précédente entre des classes du domaine ou de la cible de la propriété fusionnée, ces classes doivent elles aussi être fusionnées.

Les super-propriétés et sous-propriétés des propriétés fusionnées deviennent quant à elles respectivement super-propriétés et sous-propriétés de la nouvelle propriété.

A nouveau, si une propriété n'est pas jugée pertinente pour le but assigné à l'ontologie globale, elle peut être omise.

Cardinalités

Un autre point important qui doit être pris en compte tant pour la fusion de classes que de propriétés est la question des cardinalités. En effet, si l'on fusionne deux classes appartenant au domaine de la même propriété ou si l'on fusionne deux propriétés contenant la même classe dans

leur domaine, les cardinalités imposées pour la propriété dans les deux classes peuvent entrer en conflit. Afin de résoudre ce problème, il convient de toujours choisir l'intervalle de cardinalités le plus grand, c'est-à- dire choisir la plus petite cardinalité minimale et la plus grande cardinalité maximale.

Afin de bien couvrir le problème, plusieurs cas doivent être envisagés :

- si l'on fusionne deux classes et qu'une propriété n'est attachée qu'à une des deux classes (cas 1 dans la table 4.1), alors la cardinalité minimale de la propriété dans l'ontologie fusionnée sera de 0.
- dans le cas d'une fusion de propriétés, si une classe appartient au domaine d'une seule des propriétés (cas 2 dans la table 4.1), les cardinalités de la propriété fusionnée dans cette classe seront identiques aux cardinalités déjà associées à la propriété d'origine.
- si deux classes fusionnées dans l'ontologie globale et appartenant au domaine de deux propriétés jugées identiques sont fusionnées (cas 3 dans la table 4.1), seule la propriété fusionnée sera attachée à la classe de l'ontologie globale. Les cardinalités associées à cette nouvelle propriété pour la classe de l'ontologie globale seront le plus grand intervalle de cardinalité provenant de la combinaison des cardinalités des propriétés d'origine dans la classe d'origine.

Il est à noter que le cas de deux classes non jugées identiques, appartenant au domaine de deux propriétés non jugées identiques ne présente aucun intérêt.

Ontologies		**Opération**	**Ontologie fusionnée**
Onto A	**Onto B**		
`C1 :` `  p1 [5-10]`	`C2 :` `  p2 [6-11]`	Les classes `C1` et `C2` sont jugées identiques et sont fusionnées. Les propriétés `p1` et `p2` sont également jugées similaires et sont fusionnées	`C:` `  p [5-11]`

`C1 :`
`  p1 [4-6]`
	`C2 :`
	`  p2 [1-1]`
		Les classes `C1` et `C2` sont jugées identiques et sont fusionnées. Les propriétés `p1` et `p2` ne sont pas identiques.	`C :`
			`  p1[0-6]`
			`  p2[0-1]`

`C1 :`
`  p1 [4-6]`
	`C2 :`
	`  p2 [1-1]`
		Les classes `A` et `B` ne sont pas jugées identiques et sont fusionnées. Les propriétés `p1` et `p2` sont également identiques et fusionnées.	`C1 :`
			`  p12[4-6]`
			
			`C2 :`
			`  p12[1-1]`

TAB. 4.1: Gestion des cardinalités lors de la production d'une ontologie globale.

4.3.3 Exportation des individus

Maintenant que nous possédons la structure de l'ontologie globale, il faut également nous intéresser aux instances. En effet, les individus des ontologies d'origine doivent être exportés dans un format adapté à l'ontologie globale. Afin de réaliser cela, il est nécessaire de posséder la liste des correspondances entre les éléments de l'ontologie du site web et ceux de l'ontologie globale afin de savoir comment chaque classe et propriété de l'ontologie source doit être traduite dans l'ontologie globale. Pour cela, il est donc nécessaire, après chaque intégration d'un nouveau site, de mettre à jour les correspondances entre l'ontologie de chaque site déjà intégré et l'ontologie globale. Ceci se fait via une nouvelle phase de recherche de correspondances telle que décrite à la section 4.3.1.

4.3.4 Intégration des instances

Comme cela a déjà été mis plusieurs fois en évidence, si l'intégration de schémas est une étape très importante du processus d'intégration, l'intégration des instances l'est également. Ceci afin de connaître les instances représentant le même objet du monde réel présents sur différents sites et de combiner les valeurs de leurs propriétés lors de tout usage de l'ontologie globale. L'utilisateur peut ainsi bénéficier de l'ensemble des informations disponibles pour un objet en une seule fois, et il peut également comparer les valeurs des propriétés pour lesquelles cela s'impose (ex : le prix, si la consultation de l'ontologie a pour but d'effectuer un achat).

La manière la plus simple de savoir si des instances provenant d'origines différentes (sites) représentent le même objet du monde réel est de se baser sur la notion d'identifiants tels qu'ils existent dans les bases de données. Cependant, afin de garantir le bon fonctionnement de ce type de méthode, il faut que la valeur de l'identifiant soit identique pour le même objet quel que soit son site d'origine. Les propriétés identifiantes propres à un site (ex : un numéro de produit) ne peuvent donc être considérées comme des identifiants valides. Un identifiant tel que l'ISBN par contre permet de repérer sans problème les livres pour lesquels ce numéro est identique.

Cependant, si une telle propriété ou ensemble de propriétés n'existe pas, d'autres techniques basées sur la similarité entre les valeurs d'une propriété ou d'un groupe de propriétés devront être employées. Il faudra alors choisir ces propriétés de telle manière que si la similarité entre leurs valeurs pour deux individus est forte, la probabilité d'équivalence de ces deux individus soit également forte.

Dans cette section, nous développerons tout d'abord une méthode permettant de représenter de tels identifiants dans une ontologie OWL existante. Ensuite, nous présenterons une méthode permettant de découvrir les individus identiques d'une ontologie OWL en utilisant les identifiants définis précédemment ainsi qu'un moteur d'inférence. Enfin, nous porterons un oeil critique sur cette approche et proposerons certaines améliorations basées non plus sur une équivalence stricte entre les valeurs des propriétés mais sur leur similarité.

Représentation des identifiants en OWL

Afin de représenter les identifiants en OWL, plusieurs méthodes s'offraient à nous. La 1ère méthode, qui semblait la plus évidente, était de rendre InverseFunctional les propriétés identifiantes. En effet, si deux individus possèdent la même valeur pour une propriété déclarée InverseFunctional, alors ces individus représentent le même objet du monde réel (cf. figure 4.3). Ce type de propriétés correspond donc bien à la définition des identifiants dans le modèle ERA. Cependant, si dans cette approche il est possible de définir plusieurs identifiants indépendants l'un de l'autre pour une même classe, il est impossible de définir comme dans le modèle ERA un identifiant formé de plusieurs éléments (identifiants composés). De plus, si les principaux moteurs d'inférence disponibles (Pellet[2] et Racer[3]) repèrent une erreur de cohérence quand deux individus possèdent la même valeur pour une propriété *InverseFunctional*, ils ne déduisent pas que les deux individus sont identiques.

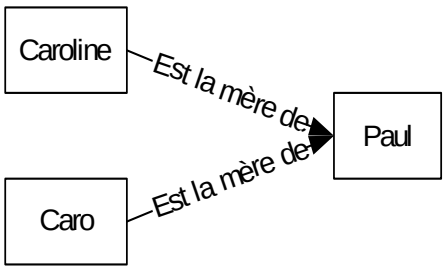


FIG. 4.3 – Exemple de propriétés InverseFunctional. Si la propriété *Est la mère de* est déclarée comme étant InverseFunctional, *Caroline* et *Caro* peuvent être considérées comme étant la même personne .

L'utilisation des propriétés *InverseFunctional* n'étant pas satisfaisante, une autre possibilité consiste à créer une propriété nommée IDProperty dont les propriétés identifiantes hériteraient. Une telle architecture demande la création non pas d'une mais de deux propriétés de ce type (une ObjectProperty et une DatatypeProperty) afin de conserver une ontologie en OWL-DL. En effet, la définition d'une ObjectProperty comme super-propriété d'une DatatypeProperty ou inversément provoque le passage de l'ontologie dans le sous-langage OWL-Full et la complétude des opérations d'inférence n'est donc plus garantie. En outre, cette approche possède différentes faiblesses. Une propriété peut posséder différentes classes dans son domaine et serait donc, selon cette méthode, identifiante pour chacune de ces classes, ce qui n'est pas nécessairement vrai. De l'information devrait donc être ajoutée afin de préciser pour quelle(s) classe(s) une propriété est identifiante. De plus, comme la méthode précédente, cette méthode ne permet pas non plus de représenter les identifiants composés.

Les précédentes réflexions nous amènent donc à nous pencher à nouveau sur la notion d'identifiant. Le fait d'être identifiante pour une propriété est propre à une classe particulière et ce de manière indépendante des autres classes auxquelles cette propriété peut être attachée. L'information concernant les identifiants doit donc être attachée à la classe et non à la propriété. Une construction définie dans l'article [Noy et al., 2000], les *own-slots*, pourrait nous permettre de représenter ce type d'informations. Pour rappel, les *own-slots* sont attachés à un objet d'une base de connaissances (classe, propriété, instance) et décrivent un attribut propre à cet objet. Les own-slots d'une classe ou d'une propriété ne sont pas hérités par les sous-classes/propriétés ni par les instances.

[2] `http://www.mindswap.org/2003/pellet/`
[3] `http://www.racer-systems.com/`

La création d'un own-slot IDProperty (de cardinalités [0,∞]) associé à chaque classe OWL de l'ontologie globale et possédant comme cible une propriété serait bien une solution répondant à nos attentes pour la définition des identifiants, ceux-ci étant comme souhaité associés à la classe. Cependant ce type de slots est propre au modèle de connaissance de l'éditeur Protégé (cf. section 2.4.4) et ne possède pas de correspondant direct dans OWL. Or la mémorisation de cette information dans l'ontologie OWL est importante pour ne pas avoir à redéfinir les identifiants à chaque recherche des instances équivalentes. En outre, si les instances sont définies dans une ontologie séparée de la définition de leurs classes, il est intéressant de pouvoir définir les identifiants sur l'ontologie de définition des classes et de pouvoir utiliser ceux-ci dans l'ontologie des instances.

Après un examen plus attentif de la définition du langage OWL, il s'est avéré qu'un type de propriétés similaire existait, les "annotations". Celles-ci sont des ObjectProperties ou DatatypeProperties (cf. section 2.4.2)qui peuvent être associées à n'importe quel élément présent dans une ontologie OWL (classe, individu, propriété,...). Ces dernières étant des constructions OWL, elles nous permettent de représenter en OWL ce que les own-slots nous permettaient de représenter dans le modèle de connaissances de Protégé. Leur utilisation comporte cependant certains petits bémols qui peuvent être gérés. En effet, comme les own-slots, ces propriétés ne sont pas héritées par les sous-classes et les identifiants doivent donc être redéfinis pour chaque sous-classe. Ensuite, les AnnotationsProperties étant soit des DatatypeProperties, soit des ObjectProperties, elles ne peuvent posséder comme cible que des types simples (nombre,chaîne de caractères,...) ou des instances de classes, mais en aucun cas une propriété. Nous mémoriserons donc leur simple nom dans une datatype annotation property et un mécanisme de gestion de l'intégrité devra être mis en place.

Les constructions permettant la définition des identifiants étant mises en place, il reste à choisir la structure de définition proprement dite. Afin de déterminer celle-ci, il est nécessaire de prendre en compte les éléments suivants :

- Un identifiant peut être une simple propriété ou être composé de plusieurs d'entre elles.
- Une classe possède zéro, un ou plusieurs identifiant(s)

De ces affirmations découle la structure choisie pour représenter les identifiants dans une ontologie OWL et représentée à la figure 4.4. De plus, afin de rendre cette structure indépendante de l'ontologie sur laquelle elle est appliquée, il est intéressant de la définir de manière externe et de l'importer dans l'ontologie voulue.

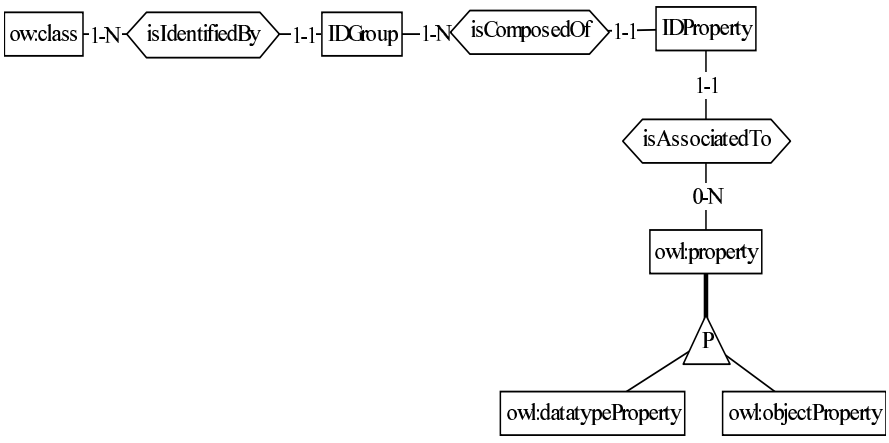


FIG. 4.4 – Structure de l'ontologie de représentation des identifiants en OWL.

Comme cela a déjà été mis en évidence précédemment, la mémorisation des propriétés identifiantes à l'aide de leur simple nom dans une DatatypeProperty nécessitera la mise en place d'un mécanisme de vérification d'intégrité. En effet, lorsqu'en OWL, une ontologie est importée dans une autre, un préfixe de *namespace* doit lui être associé afin de pouvoir faire référence à son contenu dans l'ontologie qui l'importe. Tout élément (classe, propriété, individu) de cette ontologie importée ne pourra être utilisé que si son nom est précédé du préfixe de son namespace.

Si l'on importe l'ontologie globale de la structure sur laquelle des identifiants ont été définis via la méthode décrite ci-avant, le nom des propriétés identifiantes mémorisées n'est plus suffisant pour désigner celles-ci et le préfixe associé au namespace de l'ontologie de la structure doit leur être ajouté.

Lorsqu'une propriété appartenant à au moins un groupe identifiant est supprimée ou renommée, ces groupes identifiants doivent être mis à jour afin de ne pas contenir d'information périmée.

Lorsque les deux cas précédents se présentent, différentes actions sont possibles :

- Si la propriété précédée du préfixe de l'ontologie de définition des classes existe, on peut remplacer le nom de propriété identifiante mémorisée dans la *DatatypeProperty* par sa version préfixée.
- Si la propriété ne peut être trouvée (même préfixée), une solution serait de supprimer complètement le groupe identifiant contenant la propriété.
- Deux autres actions sont également possibles, à savoir la suppression de la propriété du groupe identifiant sans modifier les autres propriétés, ou le choix d'une propriété de remplacement parmi les autres propriétés de la classe non encore présentes dans le groupe identifiant.

Méthode par équivalence des valeurs des propriétés identifiantes

Maintenant que nous disposons d'un moyen d'expression des identifiants dans une ontologie OWL, il faut nous intéresser à la découverte des individus possédant la même valeur pour ces propriétés. L'examen manuel de chaque individu un par un afin de repérer les égalités d'identifiants n'est pas une approche réaliste (à plus forte raison si le nombre d'individus et/ou de classes est élevé), car une telle démarche est particulièrement lente et sujette à erreurs. L'idée serait donc de bénéficier d'un moyen automatique et efficace de comparaison. Les moteurs d'inférence peuvent convenir pour s'acquitter d'une telle tâche car ils bénéficient d'algorithmes évolués et optimisés. Une solution permettant d'effectuer une telle opération est décrite dans [Anicic et al., 2005] et est illustrée à la figure 4.5. Elle se présente de la manière suivante : pour chaque individu (i1, i2 et i3 sur la figure 4.5) une classe temporaire le représentant est créée (respectivement I1, I2 et I3 sur la figure 4.5). Cette classe est déclarée comme sous-classe de toutes les classes auxquelles l'individu appartient. Les valeurs des propriétés identifiantes de cette classe sont restreintes à leur valeur pour l'individu en question et leurs cardinalités sont remplacées par le nombre exact d'occurences pour ce même individu. Par exemple, sur la figure 4.5, l'individu *i1* conduit à la création d'une classe *I1* sous-classe de la classe *Livre* dont les individus doivent posséder une seule valeur pour la propriété *ISBN* celle-ci devant être "XYZ". L'ensemble des restrictions se faisant à l'aide de conditions nécessaires et suffisantes (cf. classes équivalentes dans la section 2.4.2), les classes créées peuvent ensuite être soumises à un moteur d'inférence qui pourra déduire l'équivalence entre les classes (ex : I1 est équivalent à I2 sur la figure 4.5). Ces classes représentant en réalité des instances, l'équivalence entre ces dernières peut également être inférée (ex : i1 représente le même objet que i2 sur la figure 4.5).

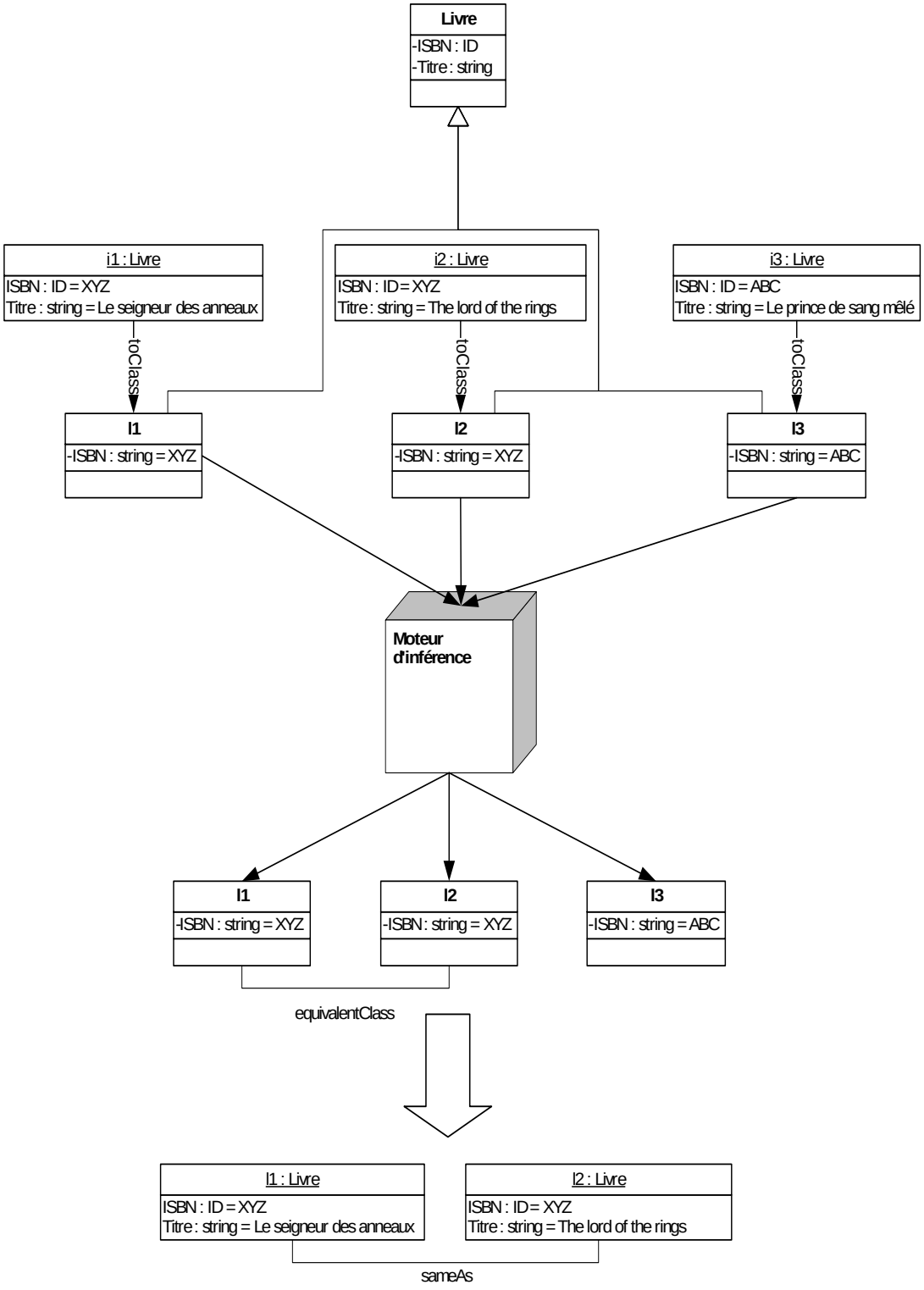


FIG. 4.5 – Exemple d'application de la méthode de déduction des instances équivalentes

Méthode par similarité des valeurs des propriétés identifiantes

Si la méthode de découverte des individus identiques basée sur l'équivalence des valeurs des propriétés identifiantes fonctionne correctement, cela n'est cependant vrai que si l'on a affaire à

un identifiant global [4]. De plus, même si les valeurs des identifiants sont censées être identiques, chaque site peut avoir représenté cette valeur de manière différente (ex : Star Wars : L'attaque des Clones → Star Wars - L'attaque des clones, utilisation d'une autre unité,...). Ensuite, une propriété facultative ne peut faire l'affaire car, si elle est présente sur un des sites et pas sur d'autres, la méthode considérera des individus identiques comme différents. Et pour cause, une propriété sans valeur n'est pas identique à une propriété valuée. Enfin, plus un identifiant contient de composants, plus la probabilité qu'une différence existe dans au moins un des composants est élevée.

Compte tenu des précédentes remarques, il s'avère que cette approche basée sur l'équivalence n'est pas suffisante et il serait donc intéressant de s'orienter vers une comparaison plus souple telle que fournie par une mesure de la similarité des valeurs. Ce type de mesure permettrait notamment d'éviter les problèmes des fautes de frappe et des identifiants globaux. En effet, ce type d'approche n'est que peu influencé par les petites différences pouvant apparaître dans les valeurs des propriétés car elle se base sur la similarité globale entre les valeurs et pas sur l'équivalence stricte entre les caractères de même position. De plus, ce type d'approche peut palier l'absence d'identifiant global : il suffit de choisir les propriétés ou combinaisons de propriétés pour lesquelles la probabilité que deux instances soient équivalentes est élevée (faible) si les valeurs de leurs propriétés choisies par un utilisateur sont hautement (faiblement) similaires.

De nombreuses approches de performances assez variables ont été développées pour mesurer la similarité entre des chaînes de caractères . Le choix de la mesure de similarité a été effectué sur la base d'une comparaison des principales méthodes effectuée dans [Cohen et al., 2003]. Les auteurs distinguent deux types principaux de mesures de similarité traditionnelles entre des chaînes de caractères.

D'une part, les algorithmes de type *distance d'édition* ou apparentés qui considèrent les chaînes de caractères telles quelles et mesurent la similarité en comptant le nombre d'opérations qui sont nécessaires pour transformer une des chaînes en l'autre chaîne. Les opérations possibles sont généralement l'insertion de caractères, la suppression de caractères et la substitution de caractères. Chacune des ces opérations se voit affecter un coût, qui sera combiné avec l'ensemble des coûts des autres opérations. Plus le coût total (la distance d'édition) sera faible, plus la similarité entre les deux chaînes sera grande.

D'autre part, les algorithmes *basés sur les tokens* qui considèrent les chaînes de caractères comme des ensembles de mots. La similarité des chaînes de caractères est alors une combinaison du nombre de mots communs entre les deux chaînes et de leur fréquence d'apparition dans chacune des chaînes.

Les auteurs proposent également un autre type de mesures qui sont produites par combinaison de techniques des deux catégories précédentes. Elle se base en général sur une technique de type *tokens* mais évaluent la correspondance entre les mots des deux chaînes non pas par simple équivalence mais à l'aide d'une mesure de similarité de la première catégorie. Ce type de mesure permet donc de gérer plus facilement les erreurs de frappe que les mesures de la seconde catégorie. La mesure *SoftTFIDF*, jugée comme la meilleure du comparatif fait partie de ce dernier type de mesures et se base sur une mesure de type token nommée *TDFIDF* et sur une mesure de distance d'édition nommée Jaro-Winkler. Ces mesures seront détaillées de manière plus approfondies dans l'annexe B.

[4]Par identifiant global, nous entendons une propriété qui possède une valeur identique quel que soit le site où l'instance est présente. Il ne faut pas les confondre avec les identifiants globaux au sens XML où la valeur littérale de l'attribut de type ID ne peut être affectée qu'à un seul élément du fichier XML.

Lorsque la similarité est calculée sur plusieurs propriétés, les mesures de similarités des différentes propriétés doivent être combinées afin d'obtenir une similarité moyenne des instances. Dans l'approche actuelle une moyenne simple est utilisée pour combiner les mesures de similarité.

Dans le cas d'une simple équivalence se basant sur des identifiants globaux, le résultat ne peut être que oui ou non. Par contre, lorsque l'on prend en compte non plus l'équivalence simple mais plutôt une mesure de similarité, le nombre de résultats possibles n'est plus fini (similarité entre 0 et 1). Ceci impose donc de définir un seuil (ou threshold) de similarité pour lequel les paires d'instances obtenant une valeur supérieure seront considérées comme équivalentes.

Ensuite, l'approbation de l'expert est indispensable afin de choisir, parmi les individus possédant une similarité suffisante (au dessus du seuil), ceux qui sont réellement identiques (ex : *Star Wars : Episode 2* et *Star Wars : Episode 3* sont deux chaînes de caractères fort similaires alors que les deux films ne le sont pas).

Représentation de l'équivalence entre deux individus dans une ontologie OWL

Un autre point sur lequel il faut s'attarder est la façon de représenter le fait que deux individus soient identiques lorsque l'expert l'a validé ou qu'un moteur d'inférence l'a détecté.

La première solution consisterait à fusionner les différents individus. Ceci paraîtrait logique vu qu'ils représentent le même objet du monde réel. Cependant, il faut noter que les valeurs des propriétés communes ne sont pas toujours les mêmes entre les différents sites d'origine. Il faudrait donc choisir quelle valeur est la plus pertinente, ceci pouvant être fait en choisissant au cas par cas ou en désignant l'un des sites comme plus fiable que les autres et donc ayant la priorité en cas de conflits.

Néanmoins, cette démarche n'est valable que si certaines des valeurs sont erronées, les informations différentes pouvant tout simplement être complémentaires (ex : être exprimées dans une autre langue), il serait donc peu judicieux de perdre certaines d'entre elles lors d'un choix. L'idée serait par conséquence de conserver toutes les valeurs pour les propriétés. Ceci est tout à fait valable si on se limite au niveau des instances. Par contre au niveau de la structure, cela implique de modifier les cardinalités afin d'autoriser davantage de valeurs pour une propriété (cardinalité maximale supérieure). De plus, la fusion de deux individus empêche de garder la spécificité de chacun et de savoir quelle information provient de quel site d'origine.

Compte tenu des faiblesses de l'approche par fusion, une autre approche doit être envisagée. Celle-ci consisterait à laisser indépendants les individus identiques mais à les lier par une construction spécifique indiquant leur équivalence. En OWL une telle construction existe : il s'agit de la construction *sameAs*.

4.3.5 Affichage des informations issues de l'intégration

Maintenant que nous possédons une ontologie représentant la structure globale de différents sites web et les données correspondantes, il s'avère intéressant d'offrir un moyen de visualisation pour celle-ci. Les structures d'origine étant des sites web, il semblerait logique de choisir de présenter l'ontologie globale sous la forme de site web.

La première question que l'on peut se poser lors de la production d'un tel "site web" est la question du contenu qu'il doit présenter. La réponse qui semble la plus évidente est l'affichage pour l'ensemble des individus de l'ontologie de l'ensemble des propriétés qui leurs sont attachées.

Ceci permet de garantir l'obtention du maximum d'informations et de toujours trouver l'information qui nous importe (ex : le prix), si du moins elle est présente dans au moins un des sites d'origine. Mais même si aucun site ne fournit l'information, on est au courant de l'absence d'information (ce qui est déjà une information).

Cependant si l'on considère que les sites sources sont assez riches en concepts et en données, l'ontologie globale contiendra elle aussi de très nombreux concepts différents et de grosses quantités de données. L'affichage simultané de cette masse d'informations ne sera pas facilement compréhensible pour un utilisateur. De plus, celui-ci ne trouvera pas aisément l'information dont il a besoin.

Un choix arbitraire de la limitation des concepts affichés ne serait pas non plus une bonne solution car on ne peut connaître d'avance l'information dont l'utilisateur aura besoin. En cas de choix maladroit, l'utilisateur ne trouvant pas l'information ne saura pas si aucun site ne la fournit ou si son absence provient d'un choix du développeur. Il serait alors obligé de parcourir chacun des sites d'origine à la recherche de son information. Ceci constituerait donc une perte de temps et le contenu de l'ontologie globale ne serait d'aucune utilité, son but original étant de rassembler l'information de différents sites afin d'éviter des consultations multiples.

Les précédentes remarques nous amènent à décider que le meilleur moyen d'assurer un compromis entre lisibilité et affichage de suffisamment d'informations est de laisser à l'utilisateur la liberté de choisir pour chaque classe de l'ontologie les propriétés qu'il souhaite voir afficher lors de sa prochaine consultation.

Un point important qui doit également être pris en compte lors de l'affichage du contenu de l'ontologie globale, est la question des instances équivalentes. Comme cela a déjà été mentionné plusieurs fois, l'intégration des individus a pour but de repérer les individus représentant un même objet du monde réel **afin d'en combiner les informations lors de toute utilisation de l'ontologie globale**. Il est donc utile, lors de l'affichage des propriétés d'un individu de l'ontologie globale, de présenter les valeurs issues de chaque site. Ceci permettant par exemple dans le cas d'un prix, de trouver le site offrant le prix le plus intéressant pour le même article.

Les ontologies fournissant des mécanismes de requêtes tout comme les bases de données, il s'avère également intéressant d'offrir une interface permettant d'exécuter de telles requêtes sur l'ontologie globale et ce de manière brute ou via une interface appropriée. Dans le cas de telles requêtes il est également important de tenir compte des équivalences entre individus. En effet, si l'on reprend l'exemple développé dans l'introduction concernant les projets et que l'on effectue une requête demandant les projets lancés avant juin 2000 et ayant un budget supérieur à € 200.000,00, aucun individu ne répondra aux critères. Effectivement la date de début est absente du site 1 et le budget n'est pas mentionné sur le site 2. Cependant, en combinant les valeurs présentes dans les deux sites pour un même projet, ce dernier pourrait alors répondre aux critères de la requête.

Une illustration de ceci peut être trouvée à la figure 4.6 où les projets ont été numérotés afin d'en faciliter la compréhension. Si l'on regarde les différents individus, aucun ne répond aux critères de la requête. Si l'on examine les valeurs des propriétés pour les différents individus, on s'aperçoit que le projet 1 a débuté avant juin 2000 et que le projet 4 a reçu un budget de plus de € 200.000,00. Or d'après la table des équivalences, ces deux individus constituent un même objet du monde réel. Cet objet correspond donc bien aux critères de la requête.

Les autres individus ne répondent pas aux critères précités. Le projet 2 ne possède pas d'équi-

valent sur l'autre site, et on n'en connaît donc pas le budget. De même le projet 3 possède une date de début de valeur inconnue. Les individus 5 et 6 sont bien identiques, leurs propriétés ne satisfont pas aux conditions imposées.

Projet								
	IDProjet	DateDeb	NomProj	Description	Budget	hasResponsable	hasMembres	lancePar
1.	PRJQRX82	8/08/1999	Proj Y	blabla	null	null	A,B,C,D	deptXZ
2.	PRJUIO34	7/05/2000	Proj Z	blabla	null	null	A,E, G, H	deptUV
3.	null	null	Proj W	blabla	250000	Harry Zona	null	null
4.	null	null	Proj Yper	blabla	210000	Sarah Fraichi	null	null
5.	PRJDRM52	2/01/2006	Proj Acces	blabla	null	null	B,R,Y,V	deptXZ
6.	null	null	Proj A.	blabla	300000	Debby Scott	null	null

SameAs	
1	4
5	6

FIG. 4.6 – Exemple de la nécessité de prendre en compte les équivalences entre instances lors de l'exécution de requêtes.

Maintenant que nous avons parcouru et développé l'ensemble de la méthode d'intégration, nous allons nous intéresser, dans le chapitre suivant, aux outils de support qui ont été utilisés pour les différentes étapes du processus.

Chapitre 5

Outils de support

Pour chacune des étapes du processus d'intégration présentées dans le chapitre précédent, des outils de support ont été utilisés lors de leur mise en pratique. Parmi ces outils certains étaient existants et ont été utilisés tels quels, d'autres ont été adaptés et d'autres encore ont été développés de a à z. Ces différents outils seront présentés individuellement dans les sections suivantes.

5.1 Retrozilla

Objectifs

Produire un XML-Schema ou une ontologie OWL représentant la structure d'un site web ainsi qu'un fichier XML ou une ontologie OWL contenant les données de ce site.

Fonctionnement

Retrozilla [Estievenart et al., 2006] est un outil d'extraction de données de sites web développé au LIBD[1] des FUNDP[2] en collaboration avec le CETIC[3], sous la forme d'un plugin au navigateur Web Mozilla. Cet outil permet de produire, à partir d'un ensemble de pages HTML d'un site (représentant le même concept), un XML-Schema ou une ontologie OWL représentant la structure de ces pages ainsi qu'un fichier XML ou une ontologie OWL contenant les données correspondantes.

Le fonctionnement global de l'outil est le suivant :

1. L'expert sélectionne un certain nombre de pages décrivant un même concept d'un site web. Ces pages doivent être bien représentatives de la diversité du site web (ex :pages où un concept est présent, d'autres où il est absent, ordre des concepts différent, nombre de valeurs d'un concept)
2. L'expert sélectionne un à un les concepts importants sur une des pages. L'outil va alors extraire les données situées au même XPath dans les autres pages. Si les données extraites des autres pages ne sont pas correctes, la sélection peut être affinée à l'aide d' un texte devant toujours précéder et/ou suivre le concept. Si un concept est multi-valué, l'expert sélectionne la première et la dernière valeur. Le nombre d'occurences est calculé automatiquement.

[1] LIBD : Laboratoire d'Ingénierie des Applications de Bases de Données
[2] FUNDP : Facultés Universitaires Notre Dame de la Paix à Namur
[3] CETIC : Centre d'Excellence en Technologies de l'Information et de la Communication

3. Un XML-Schema ou une ontologie OWL représentant la structure de la page peut ensuite être extraite ainsi qu'un fichier XML ou une ontologie OWL contenant les données.

La structure des ontologies produite est la suivante :
- Le concept représenté par les pages sera représenté par une classe.
- Les concepts constitutifs du concept principal seront reliés au concept principal
  - par une DatatypeProperty si le concept n'est pas très important et n'est pas susceptible de posséder davantage d'information que celle présentée sur les pages.
  - par une ObjectProperty si le concept est un concept important c'est-à-dire s'il est représenté par un lien hypertexte conduisant vers davantage d'informations à son propos ou si d'autres informations à son propos pourraient être découvertes ailleurs dans le site. Le concept sera alors représenté par une classe à laquelle sa valeur dans les pages sera attachée via une DatatypeProperty nommée *ConceptValue*.

Limites

La structure produite par cet outil est relativement plate (le concept de la page possède une série de sous-concepts). Cette limite est corrigée par un petit module externe qui génère une hiérarchie sur base des XPath et permet de restructurer celle-ci selon les besoins de l'expert.

5.2 PROMPT

Objectifs

Repérer les correspondances existant entre deux ontologies et produire une ontologie intégrant les concepts des deux ontologies d'origine selon les désirs de l'expert. Dans notre approche, l'ensemble des concepts et relations présents dans les ontologies de départ doivent se retrouver dans l'ontologie intégrée.

Fonctionnement

PROMPT a été développé à l'Université de Stanford (cf. [Noy and Musen, 2000, Noy and Musen, 2001, Noy and Musen, 2003]) et a été adapté lors de mon stage à l'EPFL afin de prendre en charge correctement les cardinalités OWL. Prompt est un ensemble d'outils construits sous la forme d'un plugin au framework d'édition d'ontologies Protégé (cf. section 2.4.4). Prompt permet d'effectuer différentes opérations sur les ontologies telles que la fusion de deux ontologies, la comparaison de deux versions d'une même ontologie, la division d'une ontologie en plusieurs fragments autonomes. Cet outil a été initialement prévu pour fonctionner sur la base du modèle de connaissances interne de Protégé [Noy et al., 2000] de type *frame-based*, mais s'accommode également des ontologies OWL.

La partie de l'application qui nous intéresse ici est la possibilité de fusionner des ontologies. A cette fin PROMPT suit la méthode développée dans la section 4.3.1. L'algorithme local est implémenté dans l'outil IPROMPT et l'algorithme global dans l'outil AnchorPROMPT. Ces outils permettent de repérer les correspondances entre deux ontologies et de construire une ontologie intégrant les deux autres au fur et à mesure de la validation des correspondances par l'expert. Les éléments ne possédant pas de correspondant dans l'autre ontologie peuvent être simplement copiés dans l'ontologie globale. L'outil génère en parallèle à l'ontologie globale, une autre ontologie exprimée dans le modèle de connaissance de Protégé contenant la liste des correspondances repérées entre les deux ontologies pendant le processus d'intégration. Lorsqu'il repère une correspondance, PROMPT fournit à l'expert la raison du choix de cette correspondance afin qu'il puisse plus facilement décider de la garder ou non.

Limites

Si cet outil peut théoriquement s'adapter au formalisme OWL, il éprouvait cependant certaines difficultés avec les cardinalités OWL et a donc dû être modifié afin de les prendre en charge correctement.

De plus, il commence par repérer des correspondances entre classes et propose de copier les classes ne possédant pas de correspondant dans l'autre ontologie. Dès que l'on décide de fusionner deux classes, l'outil recherche d'éventuelles correspondances entre propriétés des classes. Cependant si aucune correspondance n'est trouvée, la propriété est copiée de manière transparente. L'expert n'est donc pas averti de l'existence de cette propriété et ne peut donc pas réagir immédiatement si un correspondant non repéré par l'outil existait ou si la propriété ne devait pas être conservée dans l'ontologie globale.

Ensuite, PROMPT ne permet de repérer et d'exprimer que des correspondances simples (impliquant un seul élément de chaque ontologie), alors que dans certains cas des correspondances complexes telles que la concaténation seraient les bienvenues (ex : o1.Nom = concat(o2.Prénom, o2.Nom)).

PROMPT propose également des fonctionnalités permettant de copier les instances des classes fusionnées vers la classe correspondante de l'ontologie fusionnée, ainsi que de fusionner deux individus. Toutefois ces deux fonctionnalités posaient problème.

La première causait des pertes de valeur pour certains individus si la copie était effectuée pour toutes les instances lors de la fusion des classes. Tout fonctionnait pourtant sans problème si la copie était demandée manuellement pour une instance après exécution de la fusion. Un traitement des individus un par un n'était pas réaliste pour des grandes ontologies contenant de nombreux individus.

La deuxième quant à elle ne fonctionnait pas. Mais cela n'était pas très grave étant donné que, comme nous l'avons souligné dans la section 4.3.4, l'approche choisie ici n'est pas de fusionner les instances équivalentes mais de les lier à l'aide de la construction *sameAs*.

Enfin l'outil PROMPT dans sa version actuelle ne gére pas l'importation des ontologies OWL. Les opérations sur les instances n'étaient donc possibles que si celles-ci étaient contenues dans la même ontologie que la structure.

Ces différents problèmes ont été résolus en développant des outils spécifiques et mieux adaptés à la situation (cf. sections 5.3 et 5.4).

5.3 Instances Exporter

Objectifs

Exporter les individus présents dans les ontologies des sites web vers un format compatible avec la structure de l'ontologie globale.

Fonctionnement

L'outil PROMPT manifestant certains manquements au niveau de la gestion des instances, un autre outil a été créé par mes soins lors de mon stage à l'EPFL afin d'exporter les individus des ontologies des sites web vers l'ontologie intégrée. Pour pouvoir effectuer cette opération, il est nécessaire de posséder les ontologies contenant les données des sites web, la liste des correspondances entre la structure de l'ontologie de chaque site web et la structure de l'ontologie intégrée.

Etant donné que la structure d'un site change moins souvent que ses données, il peut s'avérer intéressant une fois que l'on possède l'ontologie intégrée de rafraîchir régulièrement les données associées. Cette opération peut être entièrement automatisée, c'est pourquoi l'outil conçu est de type batch. Il outil prend en entrée un fichier XML contenant les informations sur les différentes ontologies à utiliser. Ce fichier doit être conforme au XML Schema présenté à la figure 5.1.

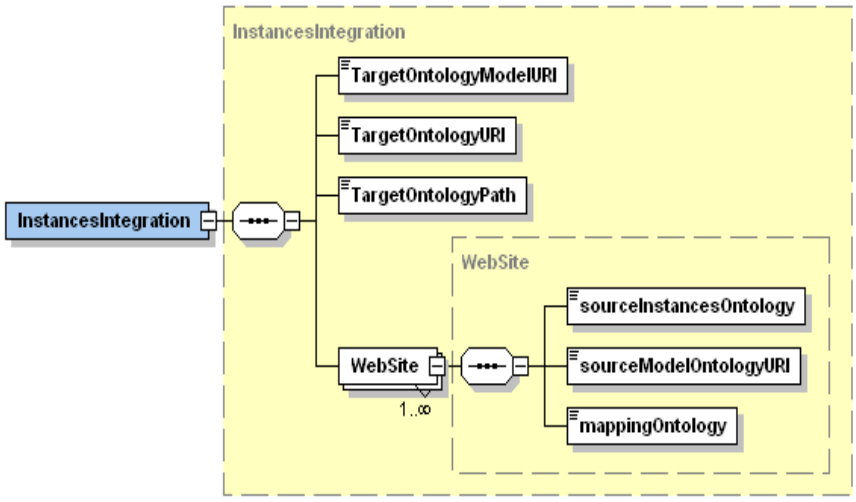


FIG. 5.1 – XML Schema auquel doit se conformer le fichier XML fourni à INSTANCES EXPORTER.

Le fichier doit tout d'abord fournir des informations concernant l'ontologie vers laquelle les instances doivent être exportées :

- L'URI de l'ontologie contenant la structure intégrée, afin de pouvoir l'importer dans l'ontologie contenant les individus que l'on va créer.
- L'URI de l'ontologie que l'on va créer et qui sera son namespace par défaut.
- Le chemin de l'emplacement où l'ontologie globale des instances doit être mémorisée.

Suit alors une série de sites web pour lesquels on fournit les informations suivantes :

- L'URI de l'ontologie contenant les instances du site web.
- L'URI de l'ontologie contenant la structure du site web.
- L'URI de l'ontologie dans le modèle de connaissance de Protégé [Noy et al., 2000] contenant les correspondances entre l'ontologie de la structure du site web et la structure de l'ontologie intégrée.

Limites

Bien que l'outil d'exportation ait été conçu pour fonctionner relativement rapidement, il n'a pas été testé sur des ontologies contenant beaucoup d'individus, et pourrait s'avérer trop peu efficace dans ce cas. Une approche de type GAV pourrait être adoptée si cela s'avérait trop lent, mais ceci provoquerait également des changements dans la découverte des individus identiques, ceux-ci n'étant plus tous localisés dans la même ontologie.

Après des mesures de durée, nous nous sommes aperçus que le chargement des ontologies en mémoire prenait beaucoup de temps par rapport au temps nécessaire à l'exportation des individus proprement dite. Le temps total pourrait sans doute être encore réduit si des améliorations du temps de chargement étaient réalisées sur l'API Protégé OWL.

Dans sa version actuelle, l'outil souffre également du même désavantage que PROMPT à savoir qu'il ne gère que des correspondances 1 élément à 1 élément, étant donné que les correspondances fournies par PROMPT sont uniquement de ce type. Cependant l'outil a été conçu pour permettre d'utiliser par la suite soit des correspondances complexes générées par PROMPT (l'ontologie exprimant les correspondances offre la possibilité d'exprimer de telles correspondances mais PROMPT ne les utilise pas) soit d'autres structures d'expression de correspondances que l'ontologie utilisée par PROMPT.

5.4 OWL-ID

Objectifs

Permettre la définition aisée d'identifiants dans une ontologie OWL, effectuer le contrôle d'intégrité des identifiants définis et permettre de découvrir les individus identiques ou similaires.

Fonctionnement

Afin de permettre de définir facilement des identifiants au sein des ontologies OWL, un plugin à l'éditeur d'ontologie Protégé a été développé par mes soins lors de mon stage à l'EPFL. Ces identifiants sont définis au moyen d'annotations (*Annotations Properties*), qui sont des *Datatype-Properties* ou *ObjectProperties* (cf. section 2.4.2), à la différence qu'une valeur de ces propriétés peut être associée non seulement aux individus mais aussi à n'importe quel élément d'une ontologie (classes, propriétés,...).

Les identifiants définis par cet outil ne peuvent être complètement considérés comme des identifiants au sens des clés primaires dans les bases de données de par le fait que deux types peuvent être définis par cet outil.

Premièrement, les identifiants pour lesquels la valeur des propriétés les constituant doit être identique afin que deux instances soient considérées comme représentant le même objet du monde réel. Ce type d'identifiant correspond à la notion existant en bases de données.

Deuxièmement les "identifiants" pour lesquels la similarité entre les valeurs des propriétés de deux instances est évaluée. Une confirmation est demandée à l'expert afin de valider si la similarité découverte entre ces deux instances justifie qu'on les considère comme représentant un même objet.

L'outil permet non seulement de définir des identifiants simples (une seule propriété) mais aussi des groupes identifiants (composés de plusieurs propriétés). Dans ce deuxième cas, les similarités entre les différentes propriétés sont combinées afin d'obtenir une similarité moyenne pour l'individu.

L'outil possède également une fonctionnalité permettant de contrôler la cohérence des identifiants définis. En effet, les annotations ne peuvent posséder comme cible que des individus ou des valeurs simples (nombres, chaînes de caractères,...) mais en aucun cas une propriété. Les noms des propriétés appartenant à un groupe identifiant doivent donc être mémorisés comme des chaînes de caractères. Ceci peut poser des problèmes de cohérence dans certains cas.

Tout d'abord, lorsque les identifiants ont été définis dans une ontologie et que l'on importe cette ontologie dans une autre, les propriétés de l'ontologie importée ne peuvent plus être utilisées

à l'aide de leur simple nom mais doivent être précédées du *namespace* par défaut de l'ontologie importée. Les groupes identifiants devraient donc être mis à jour afin de contenir ce "nouveau" nom.

Ensuite, si une propriété membre d'un groupe identifiant a été supprimée ou renommée sans que le plugin ne soit activé, les groupes identifiants risquent d'être incohérents par rapport à l'état de l'ontologie.

Il est donc important de contrôler la cohérence du contenu des groupes identifiants lors du chargement de l'ontologie dans l'outil ou lors du chargement du plugin. A cette fin, lors du lancement du plugin, un contrôle de l'ensemble des groupes identifiants est effectué. Si une propriété inconnue est découverte dans un des groupes identifiants, l'outil cherche tout d'abord si la propriété existe précédée du préfixe de la classe sur laquelle elle est définie. Si une telle propriété existe, l'outil suggère à l'expert de remplacer le nom de la propriété dans le groupe identifiant par sa version préfixée. Si une telle propriété ne peut être trouvée, l'outil propose de supprimer purement et simplement le groupe identifiant. Deux autres actions sont également possibles, à savoir la suppression de la propriété du groupe identifiant et le remplacement de celle-ci par une des propriétés attachées à la classe non encore présente dans le groupe identifiant.

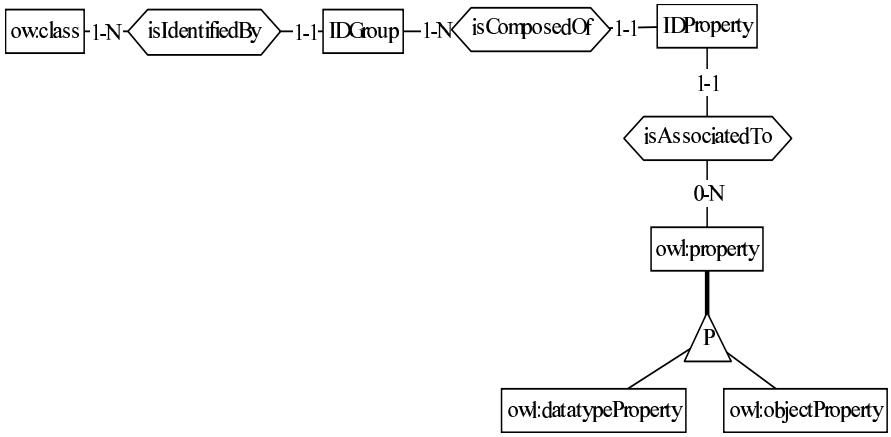


FIG. 5.2 – Structure de l'ontologie de représentation des identifiants en OWL.

Limites

Dans sa version actuelle, cet outil est seulement en mesure de calculer la similarité avec des groupes identifiants contenant uniquement des DatatypeProperties. Il n'est actuellement pas possible de définir des identifiants prenant en compte la valeur des propriétés d'un individu lié via une ObjectProperty.

Dans l'état actuel des choses, il est possible d'utiliser des ObjectProperties comme identifiants, mais il faut alors que chacun des individus identiques soit relié à un même individu via l'ObjectProperty. L'utilisation des ObjectProperties ne peut de plus se faire que pour des identifiants pour lesquels on recherche l'équivalence et non la similarité. En effet, la similarité se base sur la valeur des propriétés en tant que chaînes de caractères. Dans le cas d'une ObjectProperty, sa valeur est la valeur de la propriété *rdf :ID* de l'individu pointé par la propriété. Une similarité entre ces valeurs n'a aucun sens car le fait que deux individus possèdent un *rdf :ID* proche ne signifie absolument pas que ces individus sont similaires, sauf dans des cas exceptionnels.

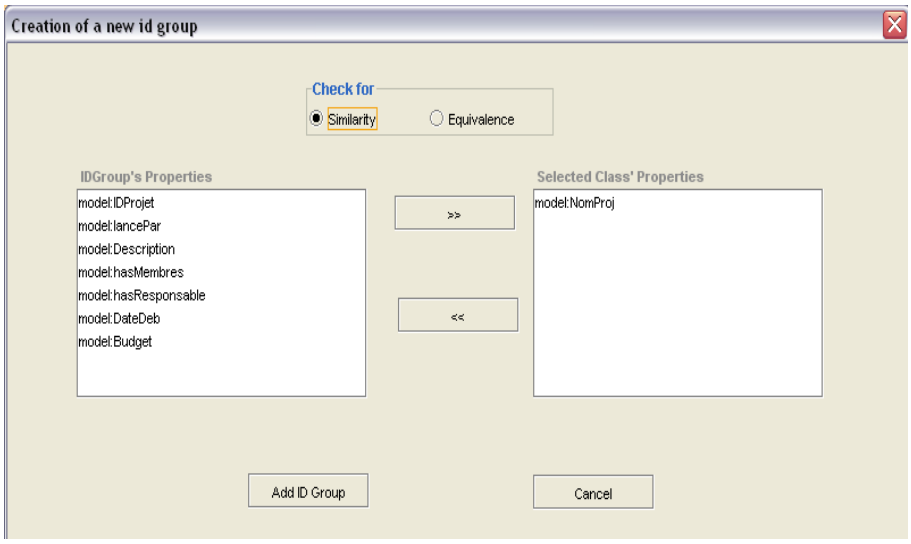


FIG. 5.3 – Ajout/édition d'un groupe identifiant permettant de choisir les propriétés le constituant mais aussi de choisir entre équivalence et similarité.

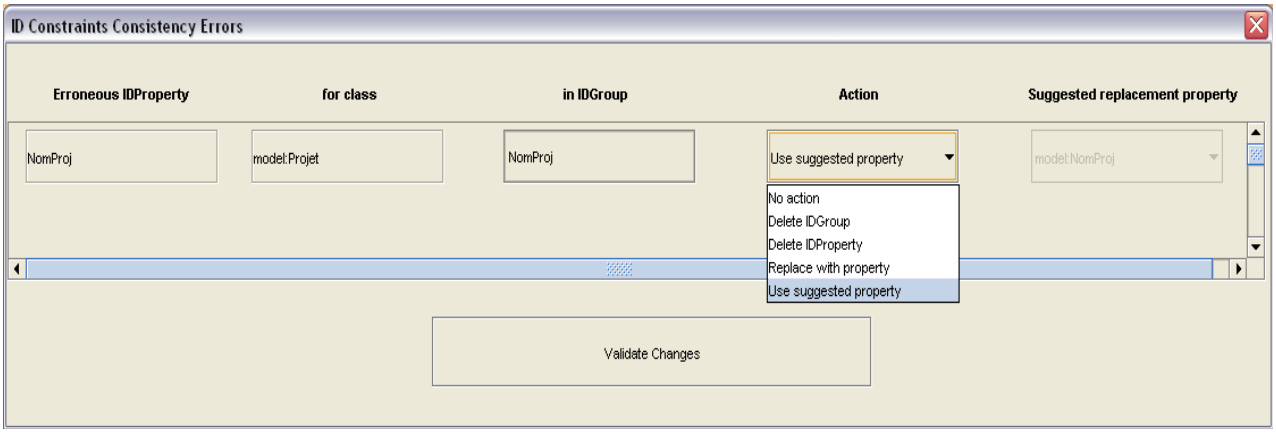


FIG. 5.4 – Fenêtre affichant les erreurs de cohérence repérées par le vérificateur de cohérence des groupes d'identifiants.

5.5 VisuWebOnto

Objectifs

Visualiser le contenu d'une ontologie OWL via une interface web.

Fonctionnement

Comme cela a déjà été mentionné à la section 4.3.5, la meilleure solution pour afficher l'ontologie globale était d'utiliser une interface web. Pour cela un ensemble de servlets Java a été créé par mes soins lors de mon stage à l'EPFL, afin de fournir une interface web au-dessus de l'ontologie globale celle-ci étant manipulée au moyen de l'API de l'éditeur Protégé.

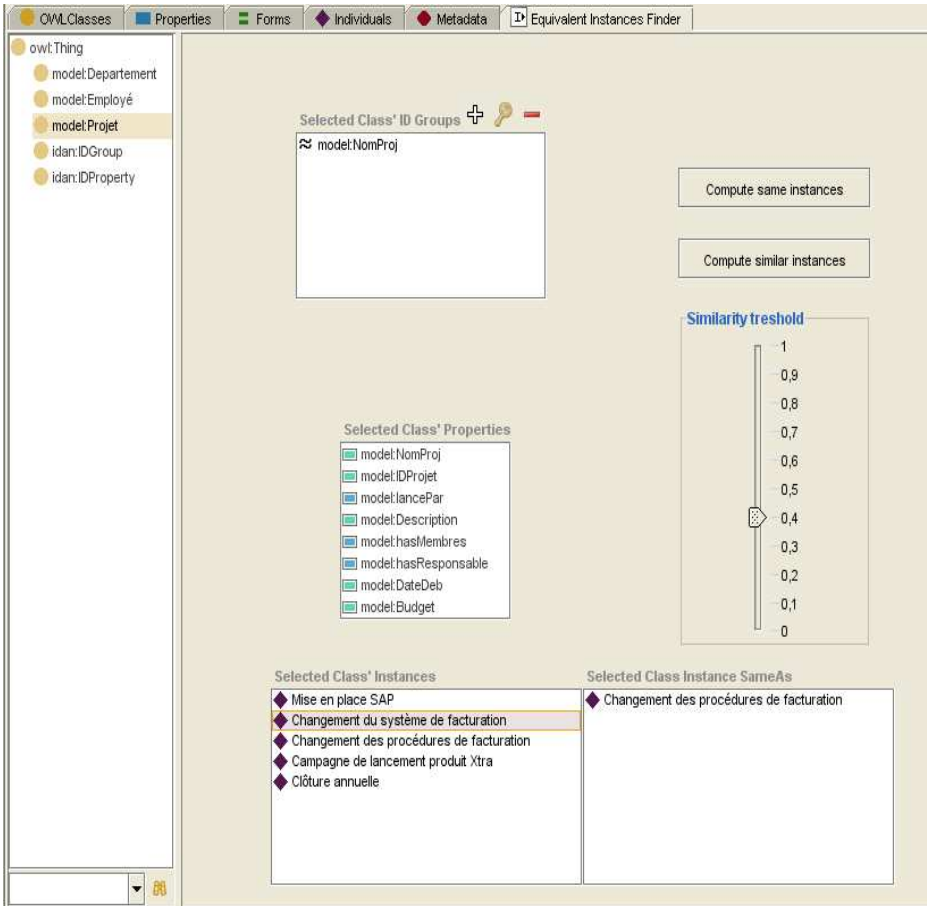


FIG. 5.5 – L'écran principal de l'application de définition des identifiants dans les ontologies OWL.

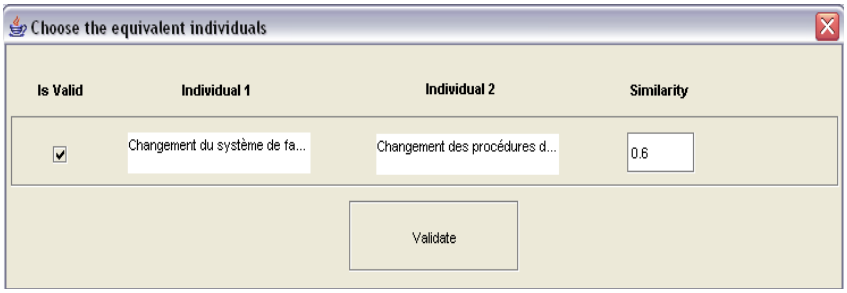


FIG. 5.6 – La demande de validation auprès de l'expert des projets dont les noms ont été considérés comme suffisamment similaires

Cette application permet à l'utilisateur de définir un profil de ce qu'il veut voir figurer sur le site web. Dans ce but, il choisit, pour chaque classe, les propriétés qu'il souhaite voir afficher pour les individus de cette classe. Lors de l'affichage d'un individu, les valeurs des propriétés de celui-ci sont affichées mais aussi celles des propriétés correspondantes pour les individus jugés comme représentant le même objet du monde réel lors de l'étape d'intégration des instances. Une gestion de plusieurs utilisateurs au moyen de logins/mots de passe a également été intégrée afin de permettre à chaque utilisateur de posséder son propre profil d'affichage de l'ontologie. Les profils sont mémorisés pour chaque ontologie visualisée à l'aide de l'outil.

L'outil propose également d'exécuter des requêtes en langage SPARQL sur l'ontologie intégrée.

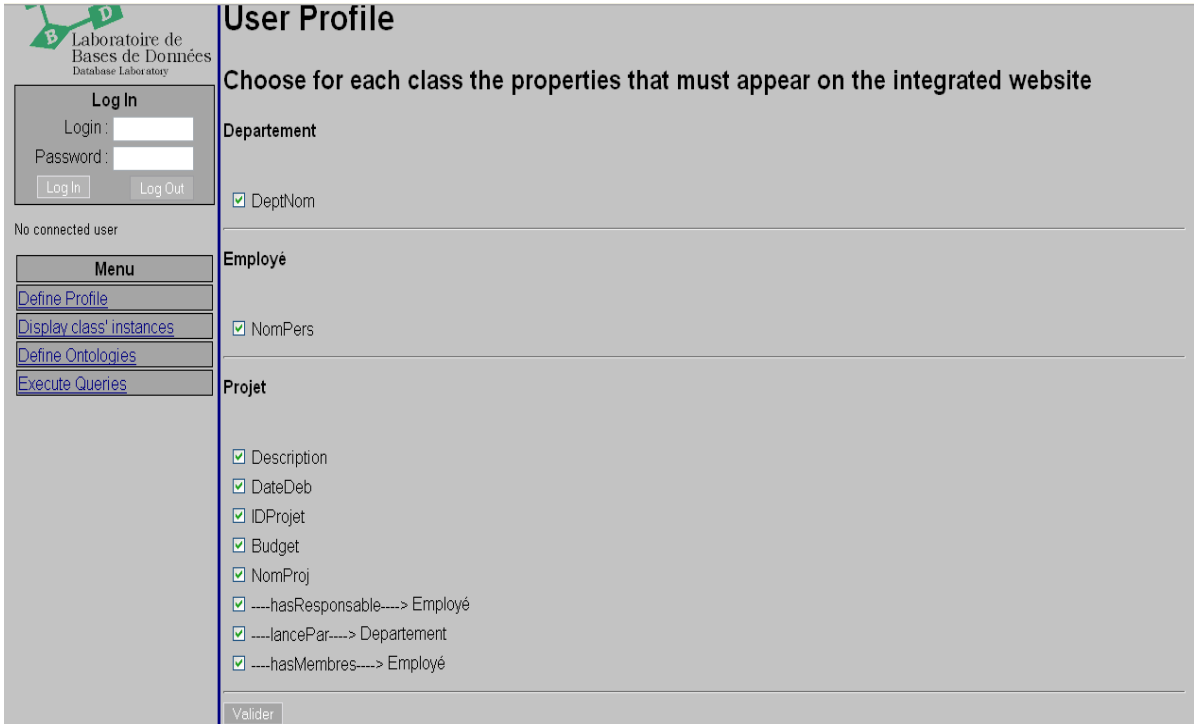


FIG. 5.7 – Définition du profil d'affichage de l'ontologie intégrée concernant les projets.

Limites

Dans sa version actuelle, le moteur d'exécution de requête de l'API OWL de Protégé est utilisé pour répondre aux requêtes soumises par l'utilisateur. Or ce moteur ne prend pas en compte la valeur des propriétés pour les individus identiques afin de décider si un individu répond aux critères de la requête. Pour pouvoir tenir compte des individus identiques comme prévu à la section 4.3.5, le moteur d'exécution de requêtes de Protégé devrait donc être amélioré.

Chapitre 6

Etude de cas

Dans ce chapitre, nous présenterons une illustration, étape par étape, de la méthode proposée au chapitre 4, appliquée à un cas concret en nous appuyant sur les outils présentés au chapitre 5.

6.1 Mise en situation

Actuellement, il existe sur Internet de nombreux sites fournissant des informations à propos de films ou DVD. Chaque site possède ses propres spécificités, certains sont des sites de vente, d'autres de simples encyclopédies du cinéma,.... Dans le cas qui nous intéresse, Monsieur X, grand amateur de films en tout genre possède une grande collection de DVD's et d'anciennes cassettes VHS et il souhaiterait avoir un inventaire informatisé de l'ensemble de ses films. Cependant il n'a pas le temps d'encoder manuellement toutes les informations et aimerait quelque chose de plus convivial qu'une simple numérisation des jaquettes. De plus, Monsieur X a l'habitude de consulter trois sites (cf. figure 6.1) sur lesquels il trouve toutes les informations qu'il souhaite. Etant familier de la consultation via Internet, il souhaiterait conserver ce type de média tout en ayant si possible à ne consulter qu'un seul site sur lequel il trouverait toute l'information nécessaire.

6.2 Description des sites utilisés

Le premier site est IMDB qui est une base de données en ligne contenant de nombreux films ainsi que des informations sur ceux-ci. Le site fournit pour chaque film les informations suivantes : le titre, le metteur en scène, l'auteur, le type de film (action, aventure, ...), les noms des différents acteurs, les grandes lignes de l'intrigue, une réplique marquante du film, une note attribuée par les internautes, les différentes certifications que le film a reçues dans différents pays, si le film est en couleur ou en noir et blanc, des commentaires, la langue, la durée, le type de bande sonore, l'année de sortie et enfin la position du film dans le classement MPAA.

Le deuxième site est DVDPlanet, un site de vente en ligne de DVD's. Ce site fournit également certaines informations concernant les différents DVD's vendus. Ces informations pour un film sont : le titre, le prix de vente normal, le prix de vente sur le site, la catégorie du film, la liste des acteurs qui y jouent, le metteur en scène, le synopsis, la position du film dans le classement MPAA, le numéro de région géographique dans laquelle le DVD est lisible, le numéro UPC, le nombre de disques, les langues disponibles pour les paroles ainsi que les sous-titres, des informations sur les caractéristiques audio et vidéo et enfin les bonus accompagnant le film.

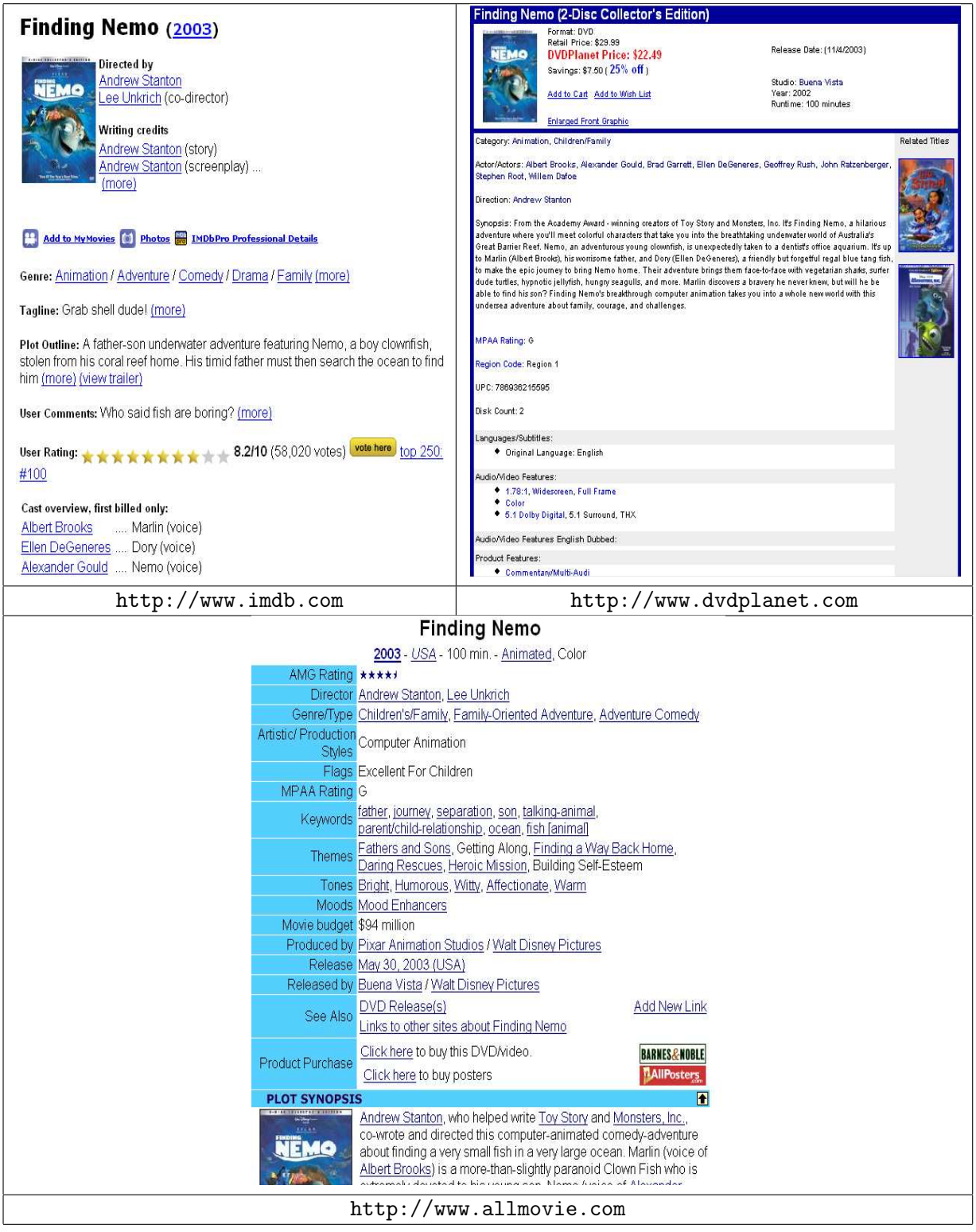


FIG. 6.1 – Les trois sites consultés régulièrement par Monsieur X

Le dernier site est AllMovies qui, comme IMDB est une base de données en ligne regroupant de l'information pour de nombreux films et DVD's. Ce site fournit comme informations pour chaque film : le metteur en scène, le genre du film, le titre, la liste des acteurs, une appréciation des auteurs du site concernant le public cible, la position du film dans le classement MPAA, un ensemble de mots-clés correspondant au film, la liste des thèmes abordés dans le film, la firme ayant produit le film, le synopsis, une critique du film, la liste des membres de l'équipe de

production, une liste de films similaires et enfin les diverses récompenses reçues par le film.

6.3 Extraction de la structure et des données

Nous prenons pour hypothèse de travail que nous possédons déjà les ontologies de la structure et des données des différents sites, extraites à l'aide de Retrozilla (cf. section 5.1).

Ces ontologies sont présentées à la figure 6.2. A nouveau, celles-ci sont représentées au moyen de schémas Entités-Associations conformément aux règles de correspondances entre les deux modèles définies à l'annexe A.

6.4 Recherche des correspondances et intégration de la structure

Pour rappel, la recherche des correspondances entre les ontologies et leur intégration est effectuée à l'aide de l'outil PROMPT (cf. section 5.2).

Lors de l'intégration de l'ontologie du premier site considéré comme le plus fiable (IMDB), aucune ontologie globale n'existe encore. L'ontologie IMDB est alors considérée comme l'ontologie globale qui sera utilisée lors de l'intégration du site suivant. Aucune recherche des correspondances n'est donc nécessaire.

Lorsque nous voulons intégrer le deuxième site (DVDPlanet), nous disposons cette fois d'une ontologie globale provenant de l'intégration du premier site et il est donc nécessaire de repérer les correspondances entre les ontologies des deux sites. A cette fin, nous chargeons les deux ontologies dans PROMPT qui va alors commencer par rechercher les similarités entre les classes d'un point de vue purement lexical dans un premier temps, n'ayant pas d'autres informations à sa disposition, vu que les deux ontologies sont encore considérées comme indépendantes. Ces premières correspondances sont présentées à la figure 6.3.
Lorsque l'on examine celles-ci, on s'aperçoit qu'il en existe trois types :

- Les correspondances correctes
- Les correspondances incorrectes
- Les correspondances manquantes

Les correspondances correctes sont les suivantes :

- Actor - Actor
- Director - Direction

La correspondance *Actor - Director* quant à elle est incorrecte. Bien que les deux noms de classes soient similaires (se terminant par "ctor"), les deux concepts ne sont pas identiques. Cette correspondance peut donc être supprimée manuellement de la liste des suggestions, mais peut également être simplement ignorée (jamais confirmée).

Des éléments auxquels il faut être très attentif lors de cette étape sont les correspondances non découvertes. Dans ce cas l'outil n'a pas repéré de similarité suffisante entre les noms des classes *Movie* et *DVDPlanet*. Cette correspondance doit donc être ajoutée manuellement (cf. figure 6.4).

Les classes *Bonus* et *Writer*, respectivement des ontologies *DVDPlanet* et *IMDB*, ne possèdent pas de correspondants dans l'autre ontologie. Celles-ci sont donc simplement copiées dans l'ontologie globale ainsi que les propriétés qui leur sont attachées.

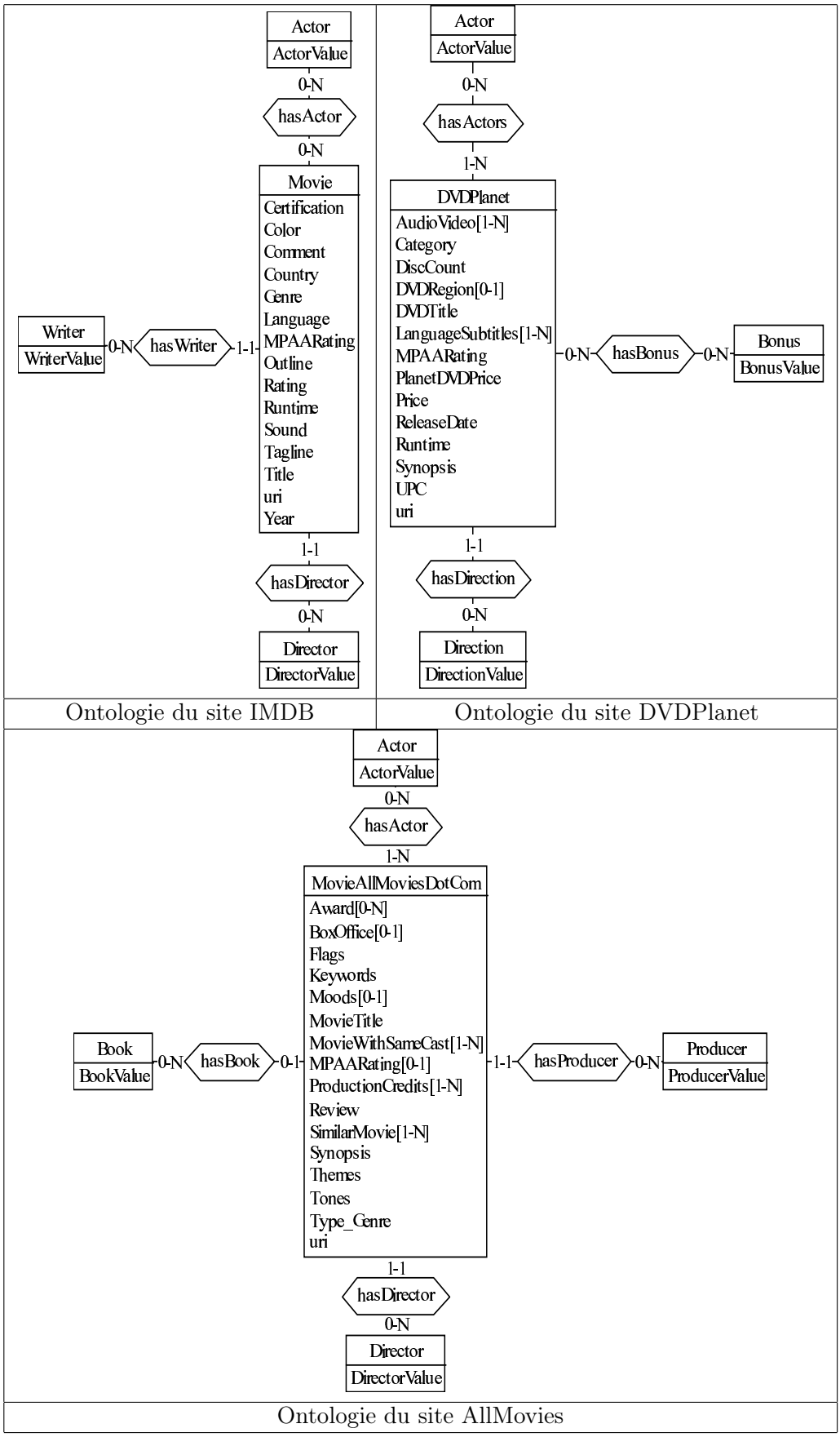


FIG. 6.2 – Les ontologies OWL représentant la structure des sites d'origine

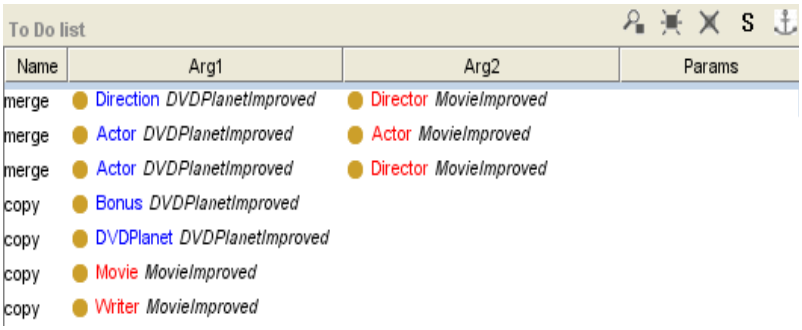


FIG. 6.3 – Les premières correspondances suggérées par PROMPT.

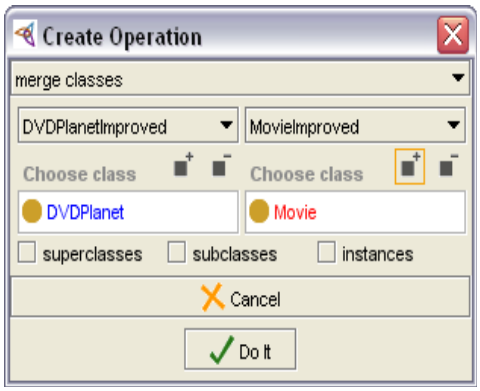


FIG. 6.4 – Ajout d'une correspondance non découverte par PROMPT entre *DVDPlanet* et *Movie*.

Lorsque l'on ajoute et valide la correspondance entre *DVDPlanet* et *Movie*, PROMPT détecte de nouvelles similarités parmi les propriétés attachées à ces deux classes. Celles-ci sont non seulement similaires lexicalement, mais possèdent également des classes identiques dans leurs domaines et sont donc présentées avant les similarités entre classes dans la liste, puisque les classes ne sont similaires que lexicalement. Cette fois toutes les correspondances ont été repérées et aucune erreur n'a été commise. Nous pouvons donc valider chacune de ces nouvelles correspondances.

Pour chaque correspondance validée, PROMPT demande de choisir le nom que l'on veut donner à l'élément intégré. Nous pourrions également décider, avant de débuter le processus de recherche de correspondances, de considérer l'une des ontologies comme plus fiable et de toujours choisir le nom venant de celle-ci. A la figure 6.6, PROMPT demande quel nom devra être employé pour la propriété résultant de la fusion de *DVDTitle* et *Title*. Dans ce cas, notre préférence se porte sur *Title*.

Lorsque toutes les correspondances valides ont été confirmées et que les correspondances absentes ont été ajoutées (cf figure 6.7), l'ontologie globale résultante est telle que présentée à la figure 6.8. Une fois que cette ontologie est complète et que l'on a intégré tout ce que l'on désirait, on peut faire appel à l'option "Remove Suffixes" afin de retirer les suffixes ajoutés par PROMPT à tout élément (ex : Title–MovieImproved indique qu'il s'agit de la propriété *Title* venant de l'ontologie du site IMDB) pour en indiquer la provenance tout au long du processus d'intégration.

Il nous faut maintenant mettre à jour les correspondances entre les deux ontologies des sites et l'ontologie globale. Pour cela, une nouvelle étape de recherche de correspondances à l'aide de PROMPT est effectuée entre chacun des sites et l'ontologie globale.

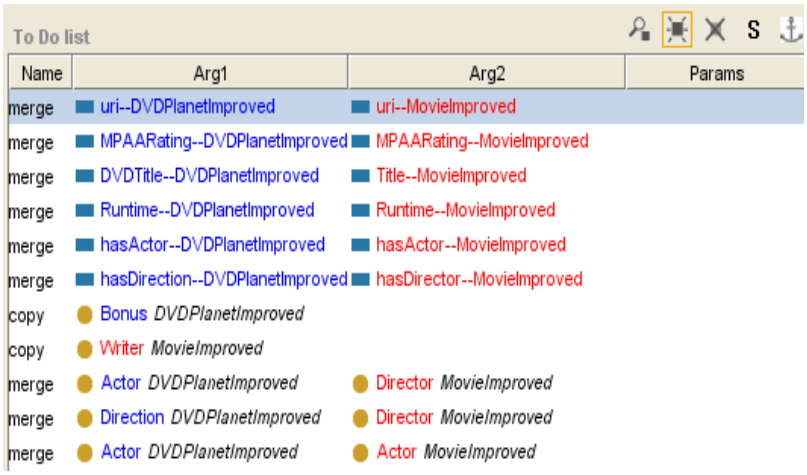


FIG. 6.5 – Les correspondances découvertes par PROMPT suite à la validation de la correspondance entre *DVDPlanet* et *Movie*

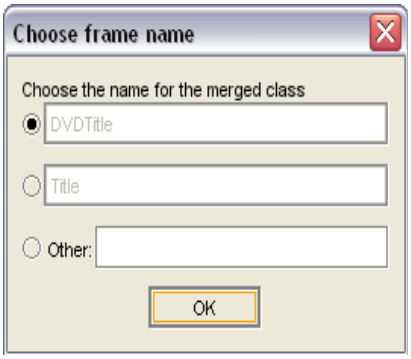


FIG. 6.6 – Choix du nom à donner à la propriété fusionnée à partir de *DVDTitle* et *Title*

Une démarche similaire est effectuée afin de produire l'ontologie globale finale à partir de l'ontologie intégrant IMDB et DVDPlanet et de l'ontologie de AllMovies. Les correspondances trouvées pendant cette étape sont présentées à la figure 6.9. L'ontologie globale finale quant à elle est représentée à la figure 6.10. Dans ce cas, l'ontologie globale produite lors de la précédente intégration est considérée comme plus fiable car regroupant déjà deux sites. De plus, des choix de noms ont déjà été faits pour certains éléments. Une fois que nous possédons l'ontologie globale, il nous faut récupérer les correspondances entre cette ontologie et l'ontologie du site AllMovie. Nous devons également, si nécessaire, mettre à jour les correspondances entre les ontologies des deux autres sites et l'ontologie globale pour prendre en compte les changements survenus sur l'ontologie lors de la deuxième intégration. Dans le cas présent, aucun changement n'a été effectué au niveau de la structure déjà existante de l'ontologie globale.

On peut remarquer que dans l'ontologie globale finale (mais aussi dans l'ontologie résultant de l'intégration des deux premiers sites), de nombreuses propriétés obligatoires sont devenues facultatives et ce parce que celles-ci ne sont pas présentes dans tous les sites. Etant donné que les individus de chaque site doivent pouvoir être stockés dans l'ontologie globale, il est nécessaire de rendre ces propriétés facultatives (cf paragraphe sur les cardinalités dans la section 4.3.1).

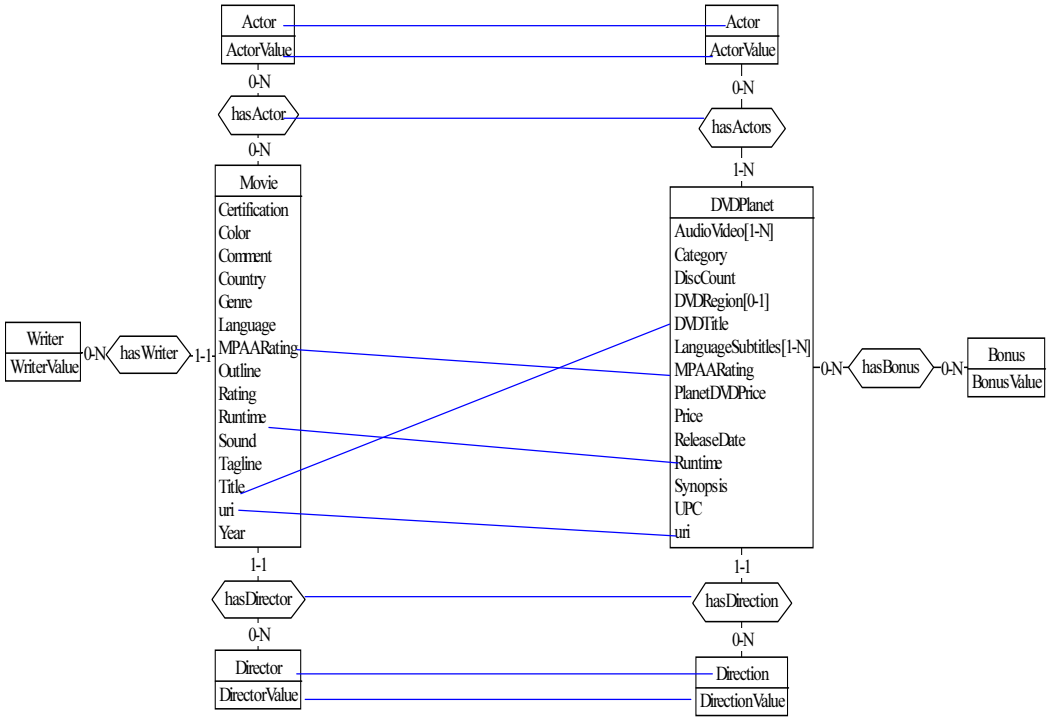


FIG. 6.7 – Les correspondances existant entre l'ontologie IMDB et l'ontologie DVDPlanet

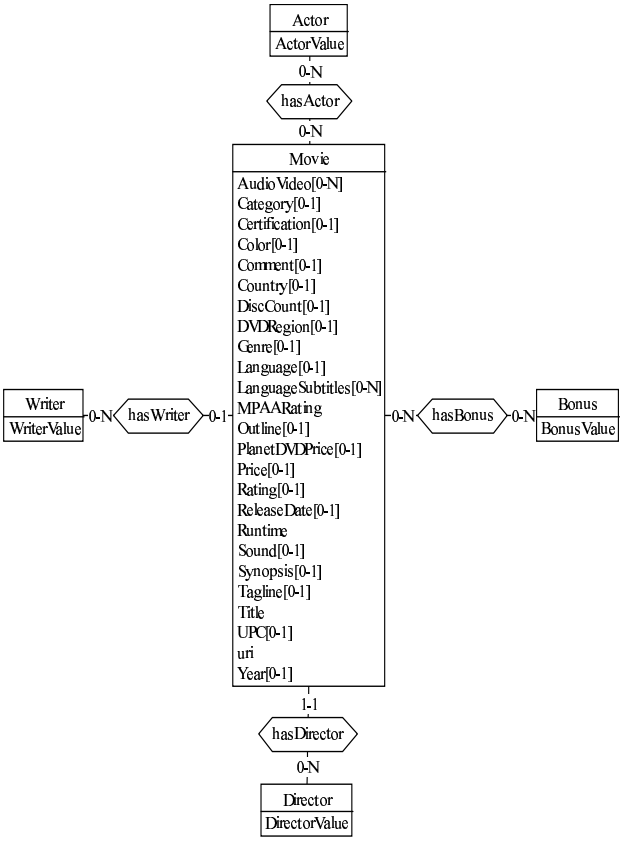


FIG. 6.8 – L'ontologie globale de la structure des sites IMDB et DVDPlanet.

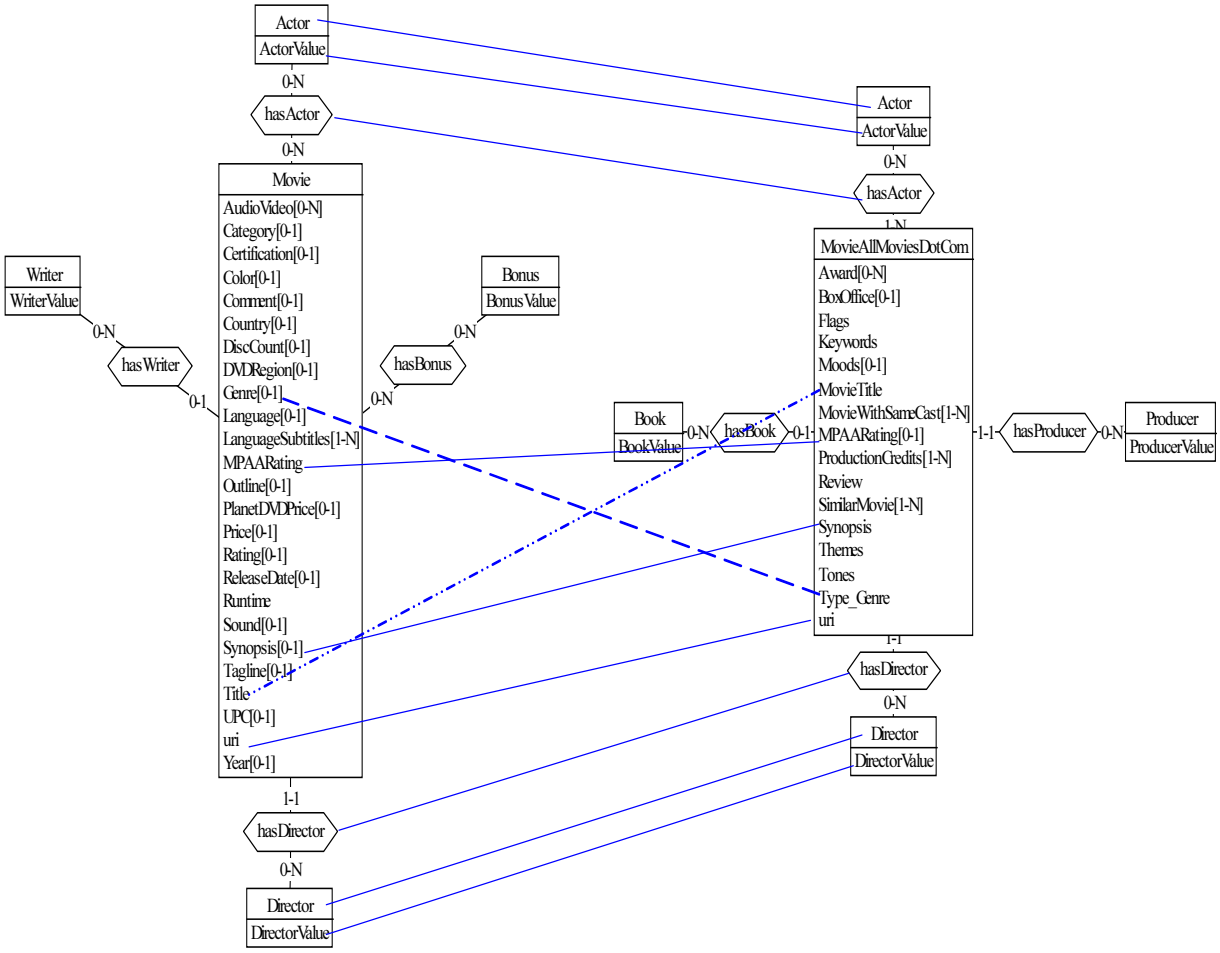


FIG. 6.9 – Les correspondances existant entre l'ontologie globale de IMDB et DVDPlanet et l'ontologie de AllMovies

6.5 Exportation des individus

Afin de réaliser l'exportation des individus à partir des ontologies des sites vers l'ontologie globale, un outil a été développé (cf. section 5.3).

Le listing 6.1 représente respectivement un individu provenant de l'ontologie du site DVDPlanet et le listing 6.2 sa traduction en un individu de l'ontologie globale lors du processus d'exportation des individus.

6.6 Intégration des individus

Nous possédons maintenant une ontologie représentant la structure globale des différents sites web, ainsi qu'une ontologie regroupant tous les individus contenant les données des différents sites dans le format de l'ontologie globale. Comme cela a été mis en évidence à la section 4.3.4, des individus représentant des mêmes objets du monde réel peuvent être présents dans les ontologies des différents sites. Afin de découvrir ces individus, nous devons tout d'abord définir des identifiants sur l'ontologie de la structure.

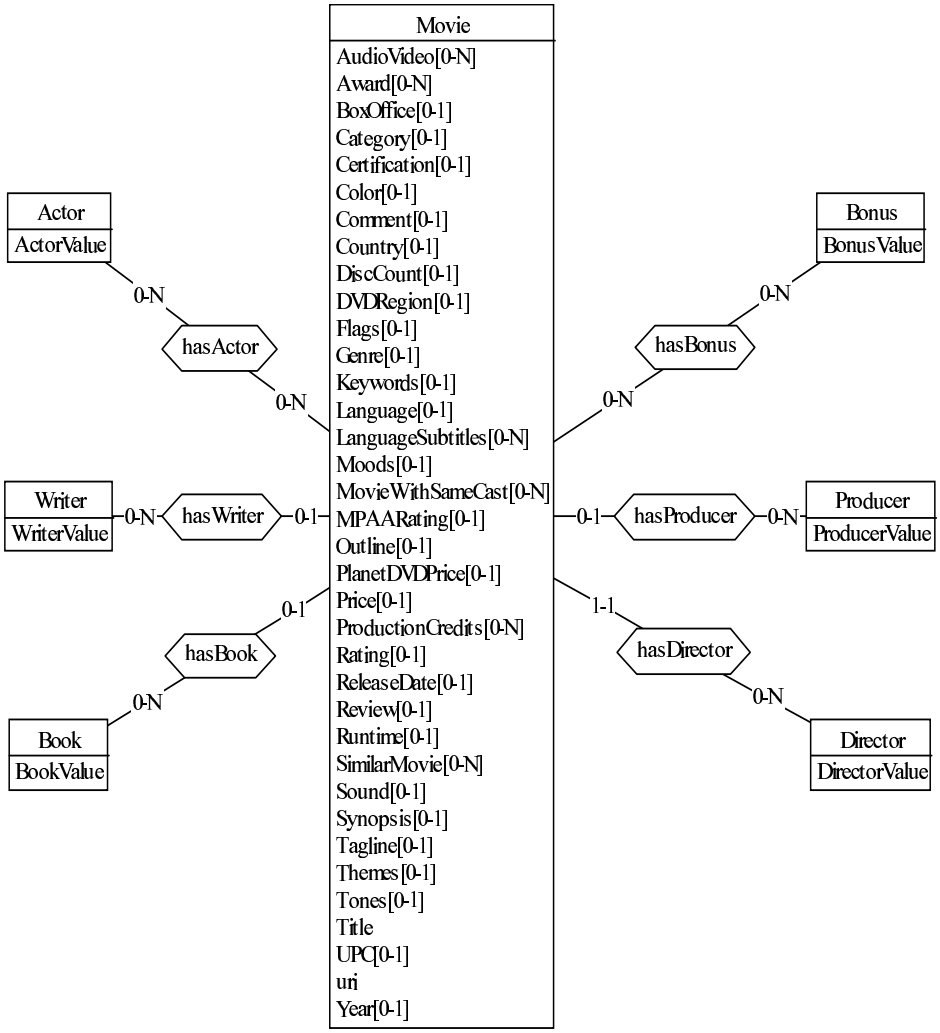


FIG. 6.10 – L'ontologie globale de la structure des trois sites d'origine.

```
<model:DVDPlanet rdf:about="file:/C:/minezilla/output/work/
    DVDPlanetImprovedInstances.owl#id132204">
        <model:DVDTitle rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                Die Another Day (Full Frame Special Edition)
        </model:DVDTitle>
        <model:UPC rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                027616888167
        </model:UPC>
        <model:DVDRegion rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                Region 1
        </model:DVDRegion>
        <model:hasActor rdf:resource="file:/C:/minezilla/output/work/
            DVDPlanetImprovedInstances.owl#id133898"/>
        <model:Runtime rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                132 minutes
        </model:Runtime>
        ...
  </model:DVDPlanet>
```

Listing 6.1 – Individu de l'ontologie du site DVDPlanet.

Dans le cas qui nous préoccupe, nous souhaitons simplement voir quels sont les films identiques dans les différents sites afin de pouvoir combiner leurs informations. Nous pourrions également

```
<model:Movie rdf:ID="Movie_186">
        <model:UPC rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                027616888167
        </model:UPC>
        <model:hasActor rdf:resource="#Actor_191"/>
        <model:Title rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                Die Another Day (Full Frame Special Edition)
        </model:Title>
        <model:Runtime rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                132 minutes
        </model:Runtime>
        <model:DVDRegion rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                Region 1
        </model:DVDRegion>
        ...
</model:Movie>
```

Listing 6.2 – Individu de l'ontologie globale.

tenter de découvrir les équivalences parmi les individus des autres classes mais cela n'aurait ici que peu d'intérêt étant donné que pour les classes autres que la classe *Movie* nous ne possédons qu'une seule propriété (ex : *DirectorValue*). Il n'y aurait donc pas d'informations intéressantes à combiner.

Il nous faut maintenant, d'une part, nous intéresser à l'identifiant que l'on pourrait utiliser pour découvrir les films identiques et, d'autre part définir, s'il vaut mieux employer un test d'équivalence des valeurs de ces identifiants ou plutôt une mesure de similarité. Si l'on examine les différentes propriétés d'un film dans l'ontologie globale, on s'aperçoit que le titre est l'attribut qui caractérise le mieux le film. En effet, peu de films possèdent le même titre et il y a donc de fortes chances que deux films de même titre soient identiques, les autres propriétés ayant moins de chance de conduire à des individus identiques. Il s'agirait par conséquence d'un bon candidat identifiant.

Cependant, si l'on examine les titres de films identiques sur les différents sites (repérés manuellement), on remarque que ceux-ci peuvent être légèrement différents selon le site (ex : *Star Wars Episode III : Revenge Of The Sith (Widescreen)* sur DVDPlanet et *Star Wars : Episode III - Revenge of the Sith* sur IMDB). L'utilisation d'un identifiant pur (avec équivalence des valeurs) n'est donc pas possible. Les films ne possédant pas exactement le même titre seront considérés comme différents. Il faudrait donc plutôt s'orienter vers une mesure de similarité. De plus, le fait que deux films aient le même titre ne signifie pas que les films sont les mêmes, puisque le titre n'est pas un identifiant universel (au sens des bases de données) d'un film. Une confirmation doit dès lors être demandée à l'expert avant de considérer de tels films comme identiques. La figure 6.11 montre la définition du groupe identifiant contenant le titre pour la classe *Movie* de l'ontologie globale.

Maintenant que l'identifiant a été défini sur l'ontologie représentant la structure globale, il peut être utilisé pour toute ontologie important cette dernière. Nous allons donc utiliser cet identifiant afin de découvrir les individus similaires parmi ceux contenus dans l'ontologie globale des données. Lors du chargement de celle-ci, le plugin de gestion des identifiants nous affiche la fenêtre représentée à la figure 6.12. L'erreur de cohérence dans cette fenêtre nous informe que la propriété *Title* définie en tant qu'"identifiant" pour la classe *model :Movie* n'a pas pu être trouvée.

En effet, lors de son importation dans l'ontologie des instances, l'ontologie contenant la structure globale a été associée à un préfixe de namespace afin de pouvoir faire référence à son contenu

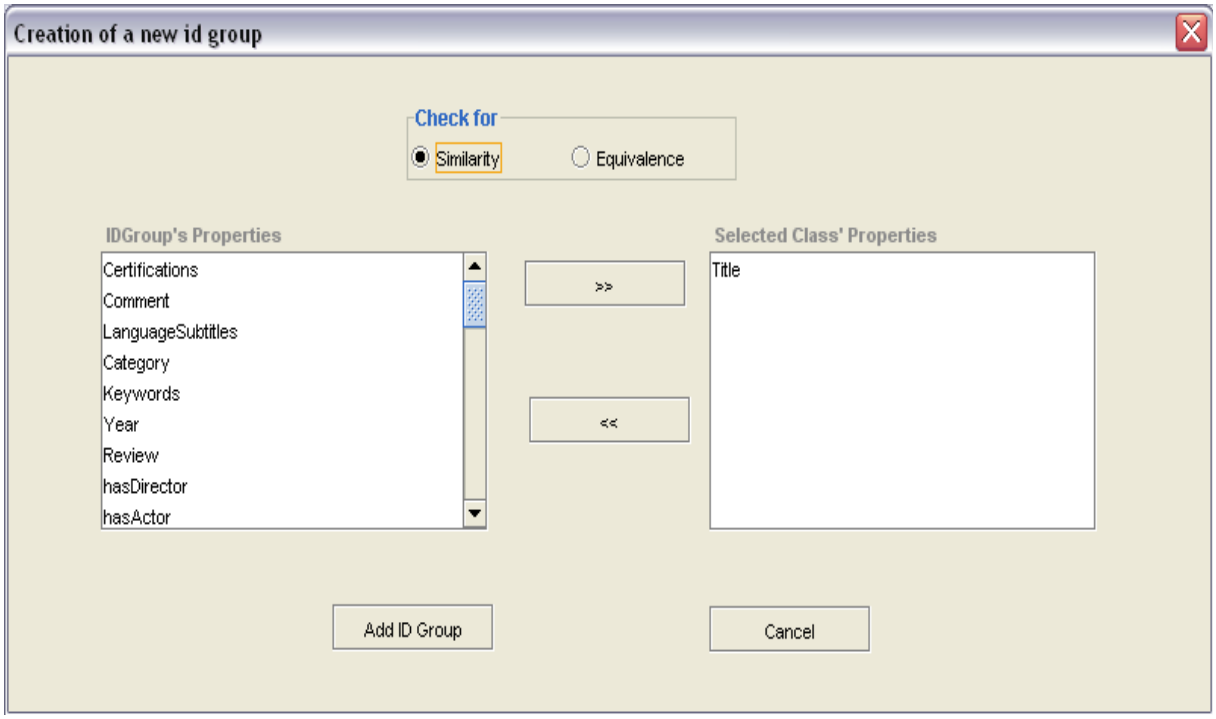


FIG. 6.11 – Définition du titre comme identifiant (par similarité) de la classe *Movie*

dans l'ontologie des instances. Les identifiants étant mémorisés dans des Annotations attachées aux classes, ces dernières n'ont aucun lien avec les propriétés et les changements opérés sur les propriétés (suppression, renommage) ne peuvent donc être automatiquement répercutés sur le contenu des identifiants.

Dans le cas qui nous occupe, si la propriété *Title* définie comme identifiant existe toujours, elle doit cependant être précédée du préfixe du namespace mentionné précédemment afin de pouvoir être référencée. Afin de connaître ce préfixe, l'outil vérifie l'existence d'une propriété de même préfixe que la classe sur laquelle l'identifiant a été défini (*model :Movie*) et de même nom que la propriété identifiante. Les identifiants étant généralement définis sur l'ontologie de la structure, il y a de fortes chances que si une telle propriété est trouvée, elle soit la propriété identifiante recherchée. Dans le cas présenté, l'outil a repéré l'existence d'une propriété nommée *model :Title*, précédée du même préfixe de namespace que la classe *model :Movie* sur laquelle l'identifiant a été défini. La propriété identifiante contenue dans le groupe identifiant (*Title*) peut donc être renommée en *model :Title*.

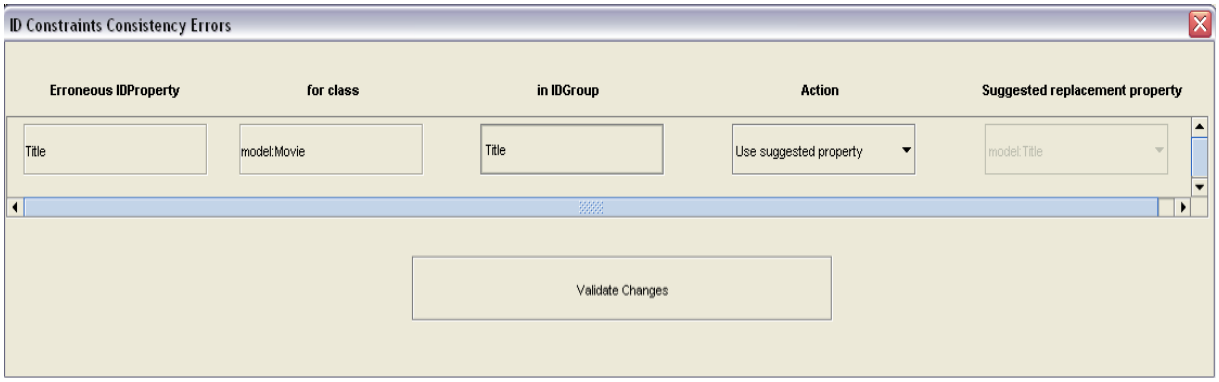


FIG. 6.12 – Erreur de cohérence des identifiants lors du chargement de l'ontologie des données.

Nous pouvons alors effectuer un calcul de similarité des différents films par rapport à leur titre. Pour ce cas, le seuil de similarité, au-dessus duquel les individus sont considérés comme suffisamment similaires pour être soumis à l'approbation de l'expert, a été positionné à 0.4. En effet, si l'on examine les titres des différents films, on s'aperçoit que bien qu'étant des films similaires les titres peuvent parfois être considérés comme peu semblables par la mesure SoftTFIDF car des informations telles que la version du film sont présentes dans le titre sur certains sites. Par exemple, *Finding Nemo* est équivalent à *Finding Nemo (2-Disc Collector's Edition)* bien que les mots communs aux deux titres soient peu nombreux par rapport au nombre total de mots. Un seuil de similarité relativement bas doit donc être choisi.

L'outil propose alors les similarités présentées à la figure 6.13. Il nous faut donc choisir parmi ces propositions, lesquelles correspondent à de réelles équivalences entre les individus. Pour ce faire il faut étudier les individus de la paire afin d'examiner les valeurs des autres propriétés. Dans ce cas, le titre des films était justement la propriété choisie dans l'éditeur Protégé afin de remplacer le simple affichage de la propriété *rdf :ID*[1] des films dans la liste des individus (n'importe quelle propriété ou ensemble de propriétés peut être choisi de la sorte) . Le titre est donc affiché dans la fenêtre d'évaluation de la similarité et ceci nous permet de déduire directement que seules quatre des propositions sont à conserver (celles-ci sont cochées sur la figure 6.13). Les autres films présents dans les autres propositions, bien que possédant des titres similaires, ne sont pas identiques. L'équivalence des différents individus est alors mémorisée dans l'ontologie contenant les données.

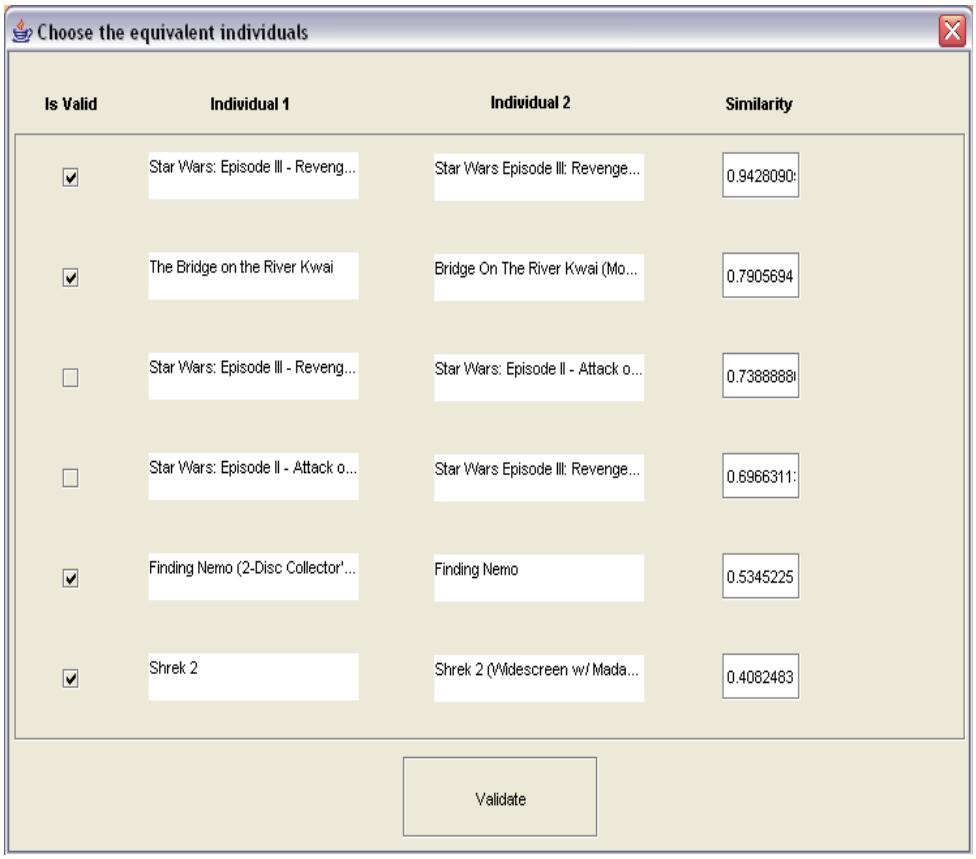


FIG. 6.13 – Les films jugés suffisamment similaires par l'outil.

[1]La propriété identifiant tout élément d'un fichier RDF ou OWL. Il s'agit d'un identifiant global au sens XML et une valeur de cette propriété ne peut donc être associée qu'à un seul élément du fichier

6.7 Affichage du contenu de l'ontologie globale

Nous disposons maintenant de tous les éléments nécessaires afin de produire un affichage du contenu de l'ontologie globale en combinant les informations des individus représentant le même objet du monde réel.

Afin de procéder à cette étape, il nous faut d'abord, si nécessaire, nous connecter (à l'aide d'un login et d'un mot de passe) sur la page d'accueil de l'afficheur d'ontologies. Ensuite, il nous faut définir notre profil d'affichage pour cette ontologie, c'est-à-dire sélectionner pour chaque classe les propriétés que l'on veut voir s'afficher. Dans notre cas, nous choisissons d'afficher les propriétés présentées dans le tableau 6.1.

Il nous faut ensuite choisir une classe parmi celles de l'ontologie, et ensuite une instance de cette classe afin d'en afficher les données. La figure 6.14 présente un exemple d'affichage d'un individu de la classe *Movie* de l'ontologie globale et possédant des équivalents parmi les autres individus. Les valeurs des propriétés *hasDirector*, *MPAARating*, *hasActor* et *Synopsis* sont affichées pour l'individu en cours mais aussi (en bleu sur le schéma) pour les individus équivalents. La propriété *Category* n'est présente que pour l'individu sélectionné et n'apparaît dans aucun des individus équivalents. Les propriétés *hasProducer* et *Keywords*, quant à elles, n'apparaissent que dans des individus équivalents et permettent donc de compléter les informations dont on dispose sur l'individu sélectionné.

Classe	**Propriétés**
Movie	hasDirector, Category, hasProducer, MPAARating, Keywords, hasActor et Synopsis
Producer	ProducerValue
Director	DirectorValue
Actor	ActorValue

TAB. 6.1 – Profil d'affichage de l'ontologie globale.

6.8 Evaluation

Nous avons démontré l'utilité de notre démarche en l'appliquant sur un cas réel. Il nous faut à présent porter un regard critique sur elle et voir quelles sont ses limites.

De manière générale, nous pouvons dire que les différents objectifs que nous nous étions fixés ont été remplis. Pour rappel, il s'agissait d'élaborer une méthode permettant d'intégrer la structure et les données de sites web et de définir une deuxième méthode permettant de découvrir les instances représentant le même objet du monde réel. Comme nous l'avons vu dans l'étude de cas, il est actuellement possible de consulter via une interface web les contenus unifiés de différents sites. Certaines améliorations pourraient toutefois être apportées à certaines étapes de la méthode ou aux outils utilisés pour ces étapes. Les limites apparaissant lors de ces différentes étapes seront présentées ci-dessous.

Recherche des correspondances et intégration de la structure

Tout d'abord, comme cela a été mis en évidence lors de la description de cette étape, la méthodologie employée ne permet actuellement de repérer que des correspondances simples (1 élément à 1 élément) alors que dans certains cas, des correspondances complexes telles que la concaténation seraient utiles afin de pouvoir tenir compte de degrés de granularité différents (ex : o1.nom = concat(o2.prénom,o2.nom)).

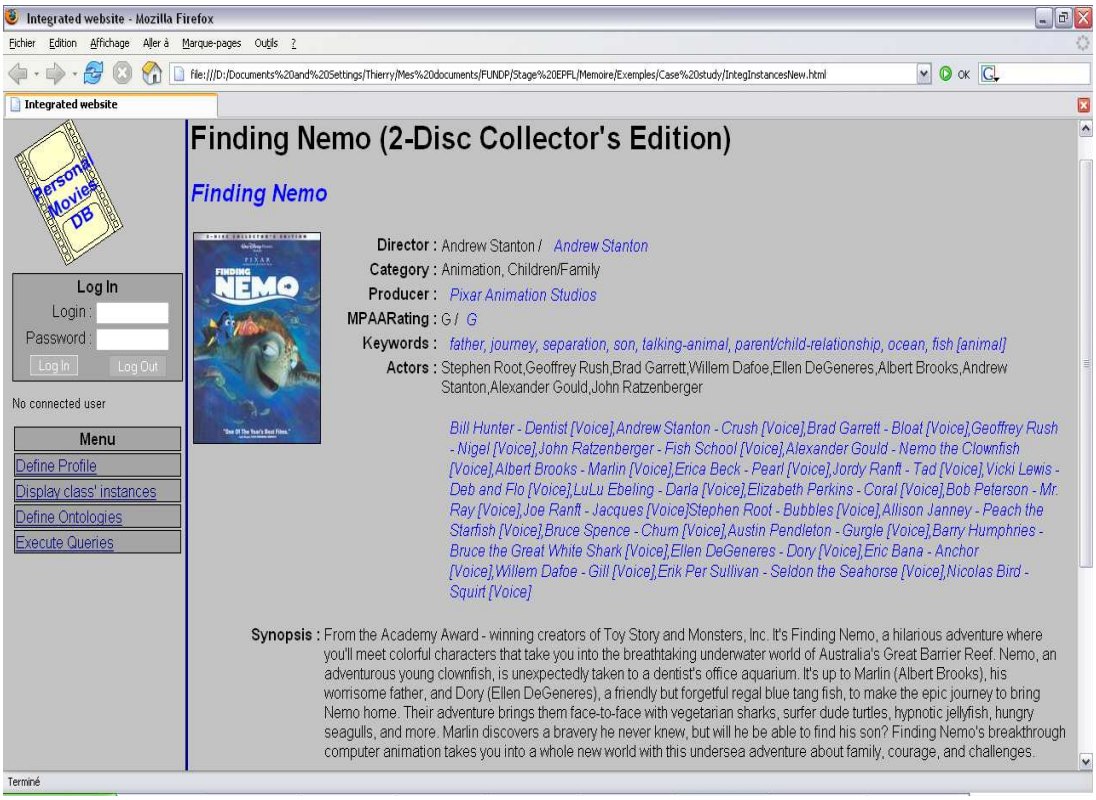


FIG. 6.14 – L'affichage d'un film de l'ontologie globale ainsi que la valeur des propriétés pour les films équivalents (en bleu).

Cette faiblesse peut cependant être contournée si on s'assure, lors de la phase d'extraction, de toujours définir les propriétés des concepts avec la granularité la plus fine possible (ex : ne pas extraire la propriété PrénomNom mais plutôt lui préférer deux propriétés distinctes Prénom et Nom). De plus, si un concept n'est présent que sous la forme d'une chaîne de caractères dans un des sites mais est susceptible de posséder davantage de propriétés dans un autre site, il est préférable de définir ce concept comme une classe reliée au concept de la page par une ObjectProperty plutôt que comme une DatatypeProperty (ex : un site fournit juste le nom du responsable d'un projet, alors qu'un autre site pourrait fournir davantage d'information quant à cette personne, comme par exemple sa date de naissance).

Ensuite, la recherche des similarités entre classes et propriétés pourrait encore être améliorée si cette similarité n'était pas calculée qu'au point de vue lexical. Il serait par exemple possible d'utiliser Wordnet[2] afin de repérer les relations intéressantes entre les noms de classes ou de propriétés (synonyme, plus général, moins général).

Exportation des individus

Bien que l'outil d'exportation ait été conçu pour fonctionner relativement rapidement, il n'a pas été testé sur des ontologies contenant beaucoup d'individus et pourrait s'avérer trop peu efficace dans ce cas. Une approche de type GAV pourrait être adoptée si cela s'avérait trop lent. Cependant, ceci provoquerait également des changements dans la découverte des individus identiques, ceux-ci n'étant plus tous localisés dans la même ontologie.

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

Après des mesures de durée, nous nous sommes aperçus que le chargement des ontologies en mémoire prenait beaucoup de temps par rapport au temps nécessaire à l'exportation des individus proprement dite. Le temps total pourrait sans doute être encore réduit si des améliorations du temps de chargement étaient réalisées dans l'API Protégé OWL.

Intégration des individus

La découverte des individus représentant le même objet du monde réel par similarité de la valeur de certaines de leurs propriétés se fait actuellement à l'aide de l'algorithme SoftTFIDF décrit comme le meilleur dans [Cohen et al., 2003]. Cependant, si par la suite, de nouvelles mesures de similarité détrônaient SoftTFIDF (précision $>$ à 0.7) et étaient applicables dans notre approche, il serait intéressant de les essayer et d'éventuellement les adopter.

Dans sa version actuelle, l'outil de gestion des identifiants et de découverte des individus identiques n'est en mesure de calculer la similarité que pour des groupes identifiants contenant uniquement des DatatypeProperties. Il n'est actuellement pas possible de définir des identifiants prenant en compte la valeur des propriétés d'un individu lié via une ObjectProperty.

Dans l'état actuel des choses, il est possible d'utiliser des ObjectProperties comme identifiants, mais il faut alors que chacun des individus identiques soit relié à un même individu via l'ObjectProperty. L'utilisation des ObjectProperties ne peut de plus se faire que pour des identifiants pour lesquels on recherche l'équivalence et non la similarité. En effet, la similarité se base sur la valeur des propriétés en tant que chaînes de caractères. Dans le cas d'une ObjectProperty, sa valeur est la valeur de la propriété *rdf :ID* de l'individu pointé par la propriété. Une similarité entre ces valeurs n'a aucun sens car le fait que deux individus possèdent un *rdf :ID* proche ne signifie absolument pas que ces individus sont similaires sauf dans des cas exceptionnels.

Affichage du contenu de l'ontologie globale

Dans la version actuelle de la méthode, l'affichage des valeurs des propriétés pour les individus équivalents à l'individu affiché est réalisé pour toutes les propriétés sans distinction. Il pourrait être intéressant de n'effectuer cet affichage que pour certaines propriétés. En effet, si l'on reprend l'exemple des films, l'affichage répété du titre n'aura guère d'intérêt pour un utilisateur car il y aura peu de différences significatives d'un site à l'autre. Par contre, l'affichage du prix pratiqué par chacun des sites sera de première importance pour un acheteur potentiel.

Afin de réaliser cela, on pourrait par exemple compléter la notion de profil utilisateur et demander à l'utilisateur de marquer dans l'ontologie globale quelles propriétés sont censées varier d'un site à l'autre. Une autre solution consisterait à introduire à nouveau une mesure de similarité afin de n'afficher la valeur d'une propriété pour un individu équivalent que si cette valeur diffère assez fort de la valeur de la propriété pour l'individu affiché.

Dans sa version actuelle, le moteur d'exécution de requête de l'API OWL de Protégé est utilisé pour répondre aux requêtes soumises par l'utilisateur. Or ce moteur ne prend pas en compte la valeur des propriétés pour les individus identiques afin de décider si un individu répond aux critères de la requête. Pour pouvoir tenir compte des individus identiques comme prévu à la section 4.3.5, le moteur d'exécution de requêtes de Protégé devrait donc être amélioré.

Chapitre 7

Conclusion

Dans ce mémoire, nous avons présenté une méthode permettant d'intégrer la structure et les données de différents sites web traitant d'une même thématique. Le résultat du processus fournit une ontologie owl décrivant la structure unifiée des différentes sources ainsi que les données exprimées dans un format conforme à cette structure. Un support logiciel pour les différentes étapes du processus a également été présenté. La phase d'extraction des données des sites web ne faisait pas à proprement parler partie des objectifs de ce mémoire. Un outil dédié à cette tâche ayant préalablement été développé par le LIBD et le CETIC, il a été emprunté et utilisé tel quel pour réaliser cette étape. Pour la phase de recherche des correspondances et pour l'intégration des structures, nous nous sommes basés sur un outil existant : Prompt (plug-in à l'éditeur d'ontologies Protégé). Après l'avoir étudié en détails, nous l'avons adapté afin qu'il réponde mieux à nos besoins spécifiques, notamment dans la gestion des cardinalités OWL. Les étapes ultérieures du processus (exportation des données vers l'ontologie globale, gestion des identifiants, découverte des instances équivalentes, affichage des informations unifiées) sont gérées par des outils entièrement développés par nos soins. L'utilité et la pertinence du processus ont été illustrés dans une étude de cas.

De manière générale nous pouvons dire que les différents objectifs que nous nous étions fixés ont été atteints. Pour rappel il s'agissait d'élaborer une méthode permettant d'intégrer la structure et les données de sites web et de définir une deuxième méthode permettant de découvrir les instances représentant le même objet du monde réel. Comme nous l'avons vu dans l'étude de cas, il est actuellement possible de consulter, via une interface web, les contenus unifiés de différents sites.

Si le présent mémoire était à refaire, je me concentrerais davantage sur une ou deux des parties du processus d'intégration afin d'obtenir une méthode sans failles pour celles-ci. En effet, dans l'état actuel des choses, un processus d'intégration complet a été spécifié et implémenté. Cependant, comme on peut le voir à la section 6.8, chacune des étapes fonctionne correctement mais possède certaines faiblesses.

Suite à ce mémoire, la piste la plus intéressante à suivre, serait de tenter d'améliorer le processus de recherche de correspondances afin de repérer des correspondances mettant en jeu plusieurs éléments d'une des ontologies comparées. En effet, la méthodologie employée ne permet actuellement de repérer que des correspondances simples (1 élément à 1 élément) alors que dans certains cas, des correspondances complexes telles que la concaténation seraient utiles afin de pouvoir tenir compte de degrés de granularité différents (ex : o1.nom = concat(o2.prénom,o2.nom)).

Une autre extension prometteuse consisterait à adapter la méthode présentée dans ce travail afin d'obtenir un système de type Global As View complet. Pour ce faire, seule la structure des

sites et la localisation de leurs données (url + localisation dans la page) devraient être extraites. Les structures pourraient alors toujours être intégrées en suivant la méthode développée dans ce document. Par contre, les données ne devraient plus être exportées vers l'ontologie globale. Les données seraient alors extraites directement lors de requêtes sur l'ontologie globale. L'étape d'intégration des individus devrait elle aussi être légèrement adaptée afin de pouvoir prendre en compte la localisation des individus dans des ontologies différentes lors de la comparaison et de la mémorisation des équivalences. De même l'étape d'affichage du contenu unifié devrait utiliser l'outil d'extraction pour accéder aux données. Une telle approche permettrait d'éviter la répétition fréquente de l'extraction des données afin de les maintenir à jour. Par contre, cette approche nécessiterait de réaliser l'étape de recherche des instances équivalentes de manière dynamique, ce qui n'est pas possible actuellement.

Enfin, lorsque nous parlons d'intégration de sites web, nous considérons en fait uniquement l'intégration des concepts et de leurs instances. Lorsque l'on examine des sites web, on s'aperçoit aisément que chaque site possède une apparence qui lui est propre. Lors de l'affichage du contenu de l'ontologie globale, l'apparence choisie actuellement n'est pas très esthétique. Or, il pourrait être intéressant d'étudier la possibilité d'intégrer des apparences de sites web, comme c'est le cas pour la structure et les données afin de produire une apparence groupant de manière optimale la présentation des données des différents sites.

Bibliographie

[Anicic et al., 2005] Anicic, N., Ivezic, N., and Jones, A. (2005). An architecture for semantic enterprise application integration standards. In *Interoperability of Enterprise Software and Applications*, Lecture Notes in Computer Science, pages 25–35. Springer.

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

[Calì et al., 2002] Calì, A., Calvanese, D., De Giacomo, G., and Lenzerini, M. (2002). On the expressive power of data integration systems. In *Proc. of the 21st Int. Conf. on Conceptual Modeling (ER 2002)*, volume 2503 of *Lecture Notes in Computer Science*, pages 338–350. Springer.

[Calvanese et al., 2001] Calvanese, D., Giacomo, G. D., and Lenzerini, M. (2001). A framework for ontology integration. In *Proceedings of the First Semantic Web Working Symposium*, pages 303–316.

[Cohen et al., 2003] Cohen, W. W., Ravikumar, P., and Fienberg, S. E. (2003). A comparison of string distance metrics for name-matching tasks. In *IIWeb*, pages 73–78.

[Doan et al., 2003] Doan, A., Lu, Y., Lee, Y., and Han, J. (2003). Object matching for information integration : A profiler-based approach. In *IIWeb*, pages 53–58.

[Doan et al., 2002] Doan, A., Madhavan, J., Domingos, P., and Halevy, A. Y. (2002). Learning to map between ontologies on the semantic web. In *WWW*, pages 662–673.

[Estievenart et al., 2006] Estievenart, F., Meurisse, J.-R., Hainaut, J.-L., and Thiran, P. (2006). Semi-automated extraction of targeted data fromweb pages. In *ICDE Workshops*, page 48.

[Fernández-Breis and Martínez-Béjar, 2002] Fernández-Breis, J. T. and Martínez-Béjar, R. (2002). A cooperative framework for integrating ontologies. *Int. J. Hum.-Comput. Stud.*, 56(6) :665–720.

[Giunchiglia et al., 2005a] Giunchiglia, F., Shvaiko, P., and Yatskevich, M. (2005a). S-match : an algorithm and an implementation of semantic matching. In Kalfoglou, Y., Schorlemmer, M., Sheth, A., Staab, S., and Uschold, M., editors, *Semantic Interoperability and Integration*, number 04391 in Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum (IBFI), Schloss Dagstuhl, Germany.

[Giunchiglia et al., 2005b] Giunchiglia, F., Shvaiko, P., and Yatskevich, M. (2005b). Semantic schema matching. In *OTM Conferences (1)*, pages 347–365.

[Gu et al., 2003] Gu, L., Baxter, R. A., Vickers, D., and Rainsford, C. (2003). Record linkage : Current practice and future directions. Technical Report 03/83, CSIRO Mathematical and Information Sciences, Canberra, Australia.

[Guha, 2004] Guha, R. (2004). Object co-identification on the semantic web.

[Hainaut, 2002] Hainaut, J.-L. (2002). Cours d'ingénierie des bases de données - partie 3 - méthodologie des bases de données. Technical report, FUNDP - Institut d'informatique.

[Horridge et al., 2004] Horridge, M., Knublauch, H., Rector, A., Stevens, R., and Wroe, C. (2004). A practical guide to building owl ontologies using the protégé-owl plugin and co-ode tools edition 1.0.

[Horrocks et al., 2003] Horrocks, I., Patel-Schneider, P., and van Harmelen, F. (2003). From SHIQ and RDF to OWL : The making of a web ontology language. *Journal of Web Semantics*, 1(1) :7–26.

[Kalfoglou and Schorlemmer, 2005] Kalfoglou, Y. and Schorlemmer, M. (2005). Ontology mapping : The state of the art. In Kalfoglou, Y., Schorlemmer, M., Sheth, A., Staab, S., and Uschold, M., editors, *Semantic Interoperability and Integration*, number 04391 in Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum (IBFI), Schloss Dagstuhl, Germany.

[Lin, 1998] Lin, D. (1998). An information-theoretic definition of similarity. In *Proc. 15th International Conf. on Machine Learning*, pages 296–304. Morgan Kaufmann, San Francisco, CA.

[Madhavan et al., 2002] Madhavan, J., Bernstein, P. A., Domingos, P., and Halevy, A. Y. (2002). Representing and reasoning about mappings between domain models. In *AAAI/IAAI*, pages 80–86.

[Maedche et al., 2002] Maedche, A., Motik, B., Silva, N., and Volz, R. (2002). Mafra - a mapping framework for distributed ontologies. In *Proc. of 13th European Conference on Knowledge Engineering and Knowledge Management (EKAW)*, Siquenca, Spain.

[McGuinness and van Harmelen, 2004] McGuinness, L. and van Harmelen, F. (2004). OWL web ontology language overview. W3C Recommendation. `http://www.w3.org/TR/2004/REC-owl-features-20040210/`.

[Mitra et al., 2000] Mitra, P., Wiederhold, G., and Kersten, M. (2000). A graph-oriented model for articulation of ontology interdependencies. *Lecture Notes in Computer Science*, 1777 :86+.

[Modica, 2002] Modica, G. (2002). *A Framework For Automatic Ontology Generation From Autonomous Web Applications*. PhD thesis, Mississippi State University, Department of Computer Science.

[Noy and Musen, 2003] Noy, N. and Musen, M. (2003). The PROMPT suite : Interactive tools for ontology merging and mapping.

[Noy et al., 2000] Noy, N. F., Fergerson, R. W., and Musen, M. A. (2000). The knowledge model of protégé-2000 : Combining interoperability and flexibility. In *EKAW*, pages 17–32.

[Noy and Musen, 2000] Noy, N. F. and Musen, M. A. (2000). PROMPT : Algorithm and tool for automated ontology merging and alignment. In *AAAI/IAAI*, pages 450–455.

[Noy and Musen, 2001] Noy, N. F. and Musen, M. A. (2001). Anchor-PROMPT : Using non-local context for semantic matching. In *Workshop on Ontologies and Information Sharing at the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001)*.

[Parent and Spaccapietra, 2000] Parent, C. and Spaccapietra, S. (2000). Database integration : The key to data interoperability. In *Advances in Object-Oriented Data Modeling*, pages 221–253.

[Prud'hommeaux and Seaborne, 2004] Prud'hommeaux, E. and Seaborne, A. (2004). SPARQL Query Language for RDF. W3C Working Draft. `http://www.w3.org/TR/rdf-sparql-query/`.

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

[Sarawagi and Bhamidipaty, 2002] Sarawagi, S. and Bhamidipaty, A. (2002). Interactive deduplication using active learning. In *KDD '02 : Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 269–278, New York, NY, USA. ACM Press.

[Shvaiko and Euzenat, 2005] Shvaiko, P. and Euzenat, J. (2005). A survey of schema-based matching approaches. In *J. Data Semantics IV*, pages 146–171.

[Smith et al., 2004] Smith, M. K., McGuinness, D. L., and Welty, C. (2004). OWL Web Ontology Language Guide. W3C Recommendation. `http://www.w3.org/TR/2004/REC-owl-features-20040210/`.

[Spaccapietra, 2005] Spaccapietra, S. (2005). Information integration. IFIP Academy,Porto Alegre.

[Stumme and Maedche, 2001] Stumme, G. and Maedche, A. (2001). Ontology merging for federated ontologies on the semantic web.

[Uschold and Gruninger, 2004] Uschold, M. and Gruninger, M. (2004). Ontologies and semantics for seamless connectivity. *SIGMOD Rec.*, 33(4) :58–64.

Annexes

Annexe A

Proposition de correspondances entre ERA et OWL

Comme nous l'avons déjà mentionné à la section 2.3, les ontologies et les bases de données possèdent des similarités au niveau de leur expressivité. Lorsque l'on examine les constructions disponibles dans les ontologies OWL et dans le modèle entités-associations (ERA), on s'aperçoit que là aussi des similarités sémantiques existent. Dans cette annexe, nous proposons un ensemble de correspondances entre ces deux modèles et verrons quelles sont les lacunes de chacun d'eux par rapport à l'autre.

Les classes

Reprenons d'abord la définition d'une *classe* dans une ontologie (voir section 2.2) :

"Une classe représente un ensemble d'objets similaires (les individus) qui partagent des caractéristiques communes. Elles peuvent également être organisées sous la forme de taxonomies (hiérarchies d'héritage)."

La définition d'un *type d'entités* dans le modèle ERA proposée par Jean-Luc Hainaut dans [Hainaut, 2002] est la suivante :

"Un type d'entités représente une classe d'objets de même catégorie (les entités) que l'on désire percevoir comme un tout : les clients (CLIENT), les commandes (COMMANDE)."

On s'aperçoit en comparant ces définitions qu'elles sont relativement semblables, les *types d'entités* étant de plus définis par analogie aux *classes*. Les *individus OWL* peuvent quant à eux être assimilés aux *entités* du modèle ERA car ils sont respectivement les composants des *classes* et des *types d'entités*, or ces deux derniers éléments se correspondent.

Les propriétés

La définition des *propriétés* de la section 2.2 nous dit que :

> "Les propriétés décrivent les caractéristiques d'une classe. Elles peuvent être vues comme des fonctions spécialisées. Elles peuvent également être organisées selon une hiérarchie et posséder des attributs particuliers (fonctionnelles, symétriques,etc.). Une propriété prend en argument un individu d'une classe nommée *domaine* et met cet individu en correspondance avec un individu d'une classe nommée *cible*."

Il est à noter que ceci correspond mieux à la définition d'une *ObjectProperty* en OWL, les *DatatypeProperties* reliant un *individu* à une valeur littérale.

Comparons cette définition avec celle des *types d'associations* donnée par Jean-Luc Hainaut dans [Hainaut, 2002] (partie 5 page 5.6)

> "Un type d'associations représente une classe d'associations de même catégorie. Les entités jouant un même rôle étant de même type."
> Une association est définie comme
> "Une association est une collection d'entités logiquement corrélées. Dans cette collection chaque entité joue un rôle spécifique."

On s'aperçoit que les *propriétés OWL* peuvent être assimilées aux *types d'associations* du modèle ERA, tous deux étant des relations reliant respectivement des *classes* et des *types d'entités* et se matérialisant par des relations entre respectivement des *individus* et des *types d'entités*. Les *types d'associations* et les *propriétés* ne sont toutefois pas complètement équivalents. En effet, un *type d'associations binaire* représente une relation bidirectionnelle entre deux *types d'entités*, chaque sens étant représenté par un *rôle*. Une *propriété* quant à elle représente une relation unidirectionelle entre deux *classes*. Les *propriétés* s'apparenteraient donc davantage aux *rôles* du modèle ERA.

Par contre, ce type de représentation n'est pas vraiment adapté pour représenter facilement une *DatatypeProperty* dans le modéle ERA. En effet, il faudrait définir un *type d'entités* représentant le type cible de la propriété et associer à ce type d'entités un attribut nommé par exemple *value* et permettant de mémoriser la valeur de la *DatatypeProperty*.

Les *DatatypeProperties* correspondraient en fait mieux à la définition d'un *attribut atomique* dans le modèle ERA. Dans [Hainaut, 2002], l'auteur définit un attribut de la manière suivante :

"Un attribut d'un type d'entités représente une propriété commune à toutes les entités de ce type. La valeur d'un attribut désigne l'état de cette propriété pour une entité particulière."

"Un attribut atomique est un attribut dont la valeur ne peut être fragmentée en composants significatifs."

"Un attribut composé est un attribut dont la valeur est constituée de valeurs plus élémentaires ayant chacune une signification précise dans le domaine d'application."

Les *ObjectProperties* pourraient également être représentées comme des *attributs composés*, étant donné l'existence d'une transformation réversible d'une entité et d'une association en *attribut composé* (cf. [Hainaut, 2002]). Un *attribut composé* provenant du modèle ERA est quant à lui obligatoirement représenté en OWL par une *ObjectProperty* et une *classe* (étant donné l'absence de *propriétés composées* en OWL)

Ajoutons qu'il n'existe pas en OWL de correspondant direct des *attributs de type d'associations*, ni des *de types d'associations n-aires* (n>2). Il est cependant possible de les représenter en les transformant d'abord en *types d'entités* (cf. [Hainaut, 2002]). Nous obtenons de cette manière un *type d'entités* et n *types d'associations binaires*, le tout étant aisément représentable en OWL.

L'héritage

Les *relations IS-A* du modèle ERA possèdent un correspondant direct en OWL, *les relations subclass-of*. En effet, ces deux relations permettent de définir des hiérarchies respectivement de *types d'entités* et de *classes*. Les *classes OWL* pouvant être assimilées à des *types d'entités*, les deux types de relations le sont également.

Les *relations d'équivalence entre classes* peuvent quant à elles être définies dans le modèle ERA à l'aide de deux *relations is-a* de sens opposé. En effet, si deux *types d'entités* sont sous-types l'un de l'autre, cela signifie qu'ils représentent le même concept.

S'il est possible en OWL de définir directement une *disjonction entre les sous-classes* d'une classe particulière, il faut cependant utiliser un mécanisme de design pattern pour décrire des *contraintes de totalité* (cf. [Horridge et al., 2004]).

Les identifiants ne font pas partie de modèle OWL sauf par définition d'une *propriété fonctionnelle inverse*. Une discussion plus approfondie à ce sujet ainsi qu'un moyen de représentation ont été proposés à la section 4.3.4.

Les cardinalités

Les cardinalités du modèle ERA peuvent être représentées de différentes façons en OWL. La manière la plus similaire consiste à utiliser les *axiomes de restriction de cardinalités* qui limitent le nombre de valeurs pour une *propriété* pour un *individu* d'une *classe*. Une autre manière plus limitée de représenter les *cardinalités* est de définir une *propriété* comme *fonctionnelle*. Dans ce cas, le nombre maximum de valeurs que pourra prendre la *propriété* pour un *individu* de n'importe quelle *classe* sera de 1. La construction *owl :someValuesFrom* ($\exists$) permet elle aussi de représenter de manière limitée les *cardinalités* en imposant une *cardinalité minimale* de 1.

Dans le modèle ERA, il est possible de définir des *cardinalités* pour chacun des *rôles* liés à un *type d'associations*. En OWL par contre il faut définir les deux *propriétés inverses* et leur associer la *cardinalité* du *rôle* correspondant.

Remarquons également que les *cardinalités par défaut* sont différentes dans les deux modèles. Dans une ontologie OWL, celles-ci sont de 0-N alors qu'elles sont de 1-1 dans le modèle ERA.

Modèle ER	**Ontologies**
Type d'entités	Classe
Type d'associations binaire	Deux ObjectProperties inverses l'une de l'autre
Type d'associations n-aire (n>2)	Une classe et 2n ObjectProperties
Type d'associations n-aire (n $\geq$ 2) avec m attributs	Une classe, m DatatypeProperties et 2n ObjectProperties
Rôles multi-TE	ObjectProperties à domaines ou cibles multiples.
Attribut d'un type d'entités	Une DatatypeProperty ou une ObjectProperty et une classe
Relation IS-A	Relation d'héritage entre classes (owl :subClass-Of)
Deux relations IS-A de sens opposé	Relation d'équivalence entre classes (owl :equivalentClass)
Cardinalités	Axiomes de restriction de cardinalités, ou de manière plus limitée l'attribut "fonctionnelle" d'une propriété et le constructeur $\exists$
Entité	Individu
Disjonction entre sous-classes	owl :disjointWith
Relation de totalité entre sous-classes	Design pattern OWL
Identifiants	cf. section 4.3.4

Tab. A.1 – Résumé des principales correspondances entre le modèle ERA et le langage OWL.

Annexe B

Mesures de similarités pour l'intégration des individus

Cette annexe présente les mesures de similarité de chaînes de caractères utilisées lors de la découverte des individus identiques à la section 4.3.4.

La mesure Jaro-Winkler

La mesure Jaro-Winkler ne fait pas vraiment partie des mesures basées sur la distance d'édition, mais comme elles, elle considère les chaînes de caractères, caractère par caractère et pas mot par mot comme les mesures basées sur les tokens. La présente méthode ne se base pas sur les opérations à effectuer pour transformer la première chaîne en la deuxième mais se base sur le nombre et l'ordre des caractères entre les deux chaînes.

Considérons deux chaînes de caractères $s = a_1 a_2 \dots a_K$ et $t = b_1 b_2 \dots b_L$. Un caractère a_i de la chaîne s est considéré comme *commun* avec la chaîne t si on peut trouver un caractère b_j de la chaîne t tel que $a_i = b_j$ et $i - H \leq j \leq i + H$ où $H = \frac{min(|s|,|t|)}{2}$.

Soit $s' = a'_1 a'_2 \dots a'_{K'}$ et $t' = b'_1 b'_2 \dots b'_{L'}$ respectivement les caractères de s communs avec t et les caractères de t communs avec s.

Une *transposition* entre s' et t' est une position i telle que $a'_i \neq b'_i$.

La mesure de similarité *Jaro* se définit comme suit :

$Jaro(s,t) = \frac{1}{3}.\left(\frac{|s'|}{|s|} + \frac{|t'|}{|t|} + \frac{|s'|-T_{s',t'}}{|s'|}\right)$ où T'_s, t' est la moitié du nombre de transpositions entre s' et t'

La mesure *Jaro-Winkler* se base sur la mesure de *Jaro* en prenant aussi en compte la longueur du plus long préfixe commun P entre s et t.

$Jaro - Winkler(s,t) = Jaro(s,t) + \frac{P'}{10}.(1 - Jaro(s,t))$
où $P' = max(P, 4)$ ce qui rend la mesure relativement spécifique aux petites chaînes de caractères.

La mesure TFIDF

La mesure TFIDF, aussi appelée Cosine Similarity, est une méthode basée sur les tokens et considère les deux chaînes de caractères s et t à comparer comme des ensembles de mots (ou tokens) S et T. Cette mesure s'intéresse aux mots appartenant à $S \cap T$, chaque mot de l'intersection recevant un poids proportionnel à sa fréquence dans les chaînes. Ce poids est élevé si peu de chaînes de l'ensemble dont S et T ont été extraites (le corpus) contiennent ce mot. Cette mesure se définit comme :

$TFIDF(S,T) = \sum_{w \in S \cap T} V(w,S).V(w,T)$ où w est un mot.

$V(w,S) = V'(w,S)/\sqrt{\sum_{w'} V'(w,S)^2}$

$V'(w,S) = log(TF_{w,S} + 1).log(IDF_w)$

où TFw, S est la fréquence du mot w dans S et IDF_w est l'inverse du nombre de chaînes du corpus qui contiennent w.

La mesure SoftTFIDF

La mesure SoftTFIDF se base sur la mesure TFIDF. Mais elle ne se base pas que sur les mots qui appartiennent à $S \cap T$ mais aussi sur ceux dont la similarité dépasse un seuil θ dans la mesure de Jaro-Winkler.

Soit $CLOSE(\theta, S, T)$, l'ensemble des mots w de S tels qu'il existe un mot v dans T tel que $Jaro - Winkler(w,v) > \theta$
Pour $w \in CLOSE(\theta, S, T), D(w,T) = max_{v \in T} Jaro - Winkler(w,v)$.

$SoftTFIDF(S,T) = \sum_{w \in CLOSE(\theta,S,T)} V(w,S).V(w,T).D(w,T)$

Annexe C

Contenu CD-ROM

Un CD-ROM est annexé à ce mémoire et contient une version pdf du présent document, le code source des différents outils employés dans la méthodologie, leur documentation et les ontologies OWL utilisées lors de l'étude de cas.

applic	Répertoire contenant les diverses applications du processus d'intégration.
bin	Le répertoire contenant les exécutables des outils de recherche de correspondances, d'intégration de la tructure, d'exportation des instances et d'intégration des instances.
EquivInstances	Répertoire contenant l'exécutable du plugin Protégé permettant de définir les identifiants et d'inférer les instances équivalentes.
InstancesExporter	Répertoire contenant l'exécutable du programme permettant d'exporter les instances des ontologies des différents sites d'origine dans une structure conforme à l'ontologie intégrée.
Prompt	Répertoire contenant le JAR de la version corrigée du plugin Prompt.
Protégé JAR	Répertoire contenant la version modifiée du fichier JAR du plugin OWL de Protégé permettant l'utilisation de " hasValue " pour les DatatypeProperties en DIG. Ainsi que les autres fichiers JAR nécessaires pour faire fonctionner l'API OWL.
SimilLib	Répertoire contenant le fichier JAR de calcul des mesures de similarité.
src	Répertoire contenant les codes sources des diverses applications du processus d'intégration.
EquivInstances	Projet Eclipse du plugin Protégé permettant de définir les identifiants et d'inférer les instances équivalentes. (Main class : gui.MainPanel)
InstancesExporter	Projet Eclipse du programme permettant d'exporter les instances des ontologies des différents sites d'origine dans une structure conforme à l'ontologie intégrée. (Main class : trt.InstancesMerger)
IntegratedWebsiteProducer	Projet Eclipse contenant les différents servlets permettant de produire le site intégré.
Prompt	Projet Eclipse de la version corrigée du plugin Prompt (Main class : edu.stanford.smi.protegex.prompt.PromptTab)

doc	Le répertoire contenant la documentation d'installation des différents logiciels du processus d'intégration.
exemples	Le répertoire contenant l'exemple sur les films utilisé dans l'étude de cas.
sites	Le répertoire contenant les pages web utilisées pour l'étude de cas.
DVDPlanet	Le répertoire contenant les pages web du site DVDPlanet utilisées pour l'étude de cas.
IMDB	Le répertoire contenant les pages web du site IMDB utilisées pour l'étude de cas.
AllMovies	Le répertoire contenant les pages web du site AllMovies utilisées pour l'étude de cas.
onto	Le répertoire contenant les ontologies utilisées pour l'étude de cas.
DVDPlanet	Le répertoire contenant les ontologies du site DVDPlanet utilisées pour l'étude de cas.
IMDB	Le répertoire contenant les ontologies du site IMDB utilisées pour l'étude de cas.
AllMovies	Le répertoire contenant les ontologies du site AllMovies utilisées pour l'étude de cas.
GlobaleTemp	Le répertoire contenant la première ontologie globale.
Globale	Le répertoire contenant l'ontologie globale finale.
GlobaleInstances	Le répertoire contenant les instances de l'ontologie globale finale.
Corresp	Le répertoire contenant les correspondances entre les ontologies locales et l'ontologie globale.
memoire	Le répertoire contenant le texte du mémoire.
images	Le répertoire contenant les diverses ilustrations utilisées dans le mémoire.
CaseStudy	Le répertoire contenant les illustrations de l'étude de cas.
utils	Le répertoire contenant des logiciels utiles pour l'application de la méthodologie d'intégration.
Pellet	Le moteur d'inférence Pellet.
Protégé	L'environnement d'édition d'ontologies Protégé.
Tomcat	Le serveur http/servlet d'Apache.
Tomcat Plugin	Le plugin Tomcat pour Eclipse.