Détection et Représentation des changements dans les sources de données RDF

Daniel Mercier[1], Nathalie Pernelle[1], Fatiha Saïs[1], Sujeeban Thuraisamy[1]

UNIVERSITÉ PARIS SUD, LABORATOIRE DE RECHERCHE EN INFORMATIQUE
91405 Orsay cedex, France
{pernelle, sais}@lri.fr, {DanielMercier,Sujeeban.Thuraisamy}@u-psud.fr

Résumé : De nombreuses sources de données RDF sont en évolution constante que ce soit au niveau des données ou du vocabulaire utilisé (ontologie). De nombreuses tâches d'intégration sont impactées par ces modifications qu'il s'agisse de synchroniser des données locales avec une source de données externe ou d'effectuer des traitements plus complexes comme le liage ou la fusion de données. Dans ce contexte, il est important de disposer d'outils permettant de détecter et de représenter ces changements de façon à ce que les tâches d'intégration impactées puissent mettre-à-jour leurs résultats sans devoir redémarrer un processus de zéro. De nombreux travaux se sont focalisés sur la détection, la représentation et le management des changements au niveau ontologique. Dans ce papier, nous présentons une approche permettant de détecter et de représenter des changements plus ou moins complexes que l'on peut détecter lorsque l'on s'intéresse aux seules données. Une première expérimentation a été menée sur différentes versions de DBPedia.

Mots-clés : Ontologies, Représentation des connaissances, Evolution des données et des connaissances

1 Introduction

Une des caractéristiques intrinsèque du Web des données (LOD) est la dynamicité et l'évolution permanente de son contenu. En effet, de nombreuses modifications sont apportées quotidiennement sur les données et les vocabulaires publiés sur le LOD. En 2012 (Käfer *et al.* (2012)), 76% des documents RDF du LOD ont subi au moins une modification. Ces modifications peuvent être de différents types (ajout, suppression et renommage) et peuvent intervenir à différents niveaux : au niveau ontologique (classes, propriétés, axiomes et liens avec les autres ontologies) et au niveau données (les entités, les propriétés, les valeurs des propriétés et les liens entre entités). De nombreuses tâches d'intégration de données sont impactées par ces modifications qu'il s'agisse de synchroniser des données locales avec une source de données externe ou d'effectuer des traitements plus complexes comme le liage ou la fusion de données.

Dans ce contexte, il est important de disposer d'outils permettant de détecter et de représenter ces changements de façon à ce que les tâches d'intégration impactées puissent mettre à jour leurs résultats sans devoir ré-appliquer tout le processus sur toutes les données. Par ailleurs, compte tenu du volume important des données, les tâches d'intégration de données doivent être complètement automatiques. Par conséquent, pour avoir des résultats sûrs il est parfois important de faire appel à des experts humains pour valider les résultats. Pour éviter de solliciter de tels experts chaque fois que les données subissent des modifications, il faut pouvoir identifier les résultats qui restent valides malgré les mises-à-jour. De nombreux travaux se sont focalisés sur la détection, la représentation et la gestion des changements au niveau ontologique (voir F. Zablith et al (Zablith *et al.* (2015)) pour une vue d'ensemble). Dans ce type de travaux, le problème consiste à détecter et à représenter les changements au niveau conceptuel entre différentes

versions d'une même ontologie et de s'intéresser à la cohérence de cette dernière. Certains travaux (Dinh *et al.* (2014)) se sont focalisés sur la représentation des évolutions subies par une ontologie et sur l'adaptation d'un ensemble de correspondances (mappings) entre ontologies associées à des bases de connaissances. Quelques travaux récents (Papavasileiou *et al.* (2013)) se sont intéressés au problème de détection de changement dans les données et de leur représentation par des types plus abstraits. Cependant, ces derniers ne sont pas représentés dans une ontologie.

Dans cet article, nous présentons une approche permettant de détecter et de représenter des changements plus ou moins complexes que l'on peut identifier lorsque l'on s'intéresse aux seules données. Une ontologie nommée O^{DE} a été conçue pour représenter les changements subis par les données de façon plus expressive qu'une simple liste d'ajouts et de suppressions. Le but de cette ontologie est qu'elle puisse être facilement exploitée par des tâches d'intégration de données telles que le liage de données, la découverte automatique de clés et la fusion de données. Une approche permettant de peupler cette ontologie automatiquement à partir de deux versions d'une source de données RDF a été développée.

Une première expérimentation a été menée sur différentes versions de DBPedia, une ancienne et une récente et deux versions consécutives de DBPedia.

Dans la section 2, nous présentons l'approche de détection et de représentation de l'évolution des sources de données RDF. Nous présentons ensuite en section 3 quelques exemples de requêtes permettant d'exploiter l'ontologie O^{DE}. Nous présentons en section 4 les premiers résultats d'expérimentation sur les données de DBPedia. Enfin, nous concluons et dressons quelques perspectives en section 5.

2 Approche de détection et de représentation de l'évolution de sources de données RDF

Dans cette section nous présentons notre approche de détection et de représentation de changements dans deux versions d'une source de données.

Comme nous le montrons en Figure 1, étant données deux versions old et new d'une source de données, l'outil charge le contenu de chacune des versions dans un TDB –Jena Triples Database (Owens *et al.* (2008)). Ainsi, nous obtenons deux bases de données TDB_{old} et TDB_{new} correspondant respectivement au contenu de la version old et de la version new de la source considérée. Ensuite, une méthode qui permet de détecter les changements élémentaires entre deux versions d'une source de données RDF est appliquée. Plus précisément, cette méthode prend en entrée TDB_{old} et TDB_{new} et construit deux ensembles de triplets : l'ensemble des triplets de TDB_{new} qui n'existaient pas dans TDB_{old} (triplets ajoutés) et l'ensemble des triplets de TDB_{old} qui n'existent plus dans TDB_{new} (triplets supprimés).
A partir de ces ensembles de triplets ajoutés et supprimés, une étape de représentation sémantique des changements est réalisée. En effet, grâce à l'ontologie O^{DE} (voir la sous-section 2.1) que nous avons conçue, il est possible de représenter sémantiquement les types de changements survenus entre deux versions différentes d'une source de données. Au lieu de représenter les changements uniquement par deux ensembles d'assertions ajoutées et supprimées, nous avons défini des types de changements, et chaque type est représenté par une classe de l'ontologie O^{DE} (ajout ou suppression d'une propriété, enrichissement/apauvrissement de la description des instances, ...).

Dans cette étape de représentation sémantique des changements, il s'agit, d'une part,

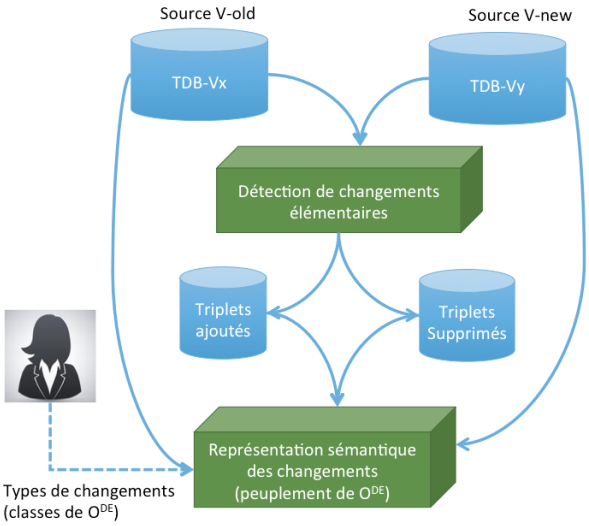

FIGURE 1 – Schéma global de l'approche de détection et de représentation de l'évolution de sources RDF

d'analyser les triplets pour détecter un ensemble de changements mais également garder trace des triplets qui sont à l'origine du type de changement détecté. Aussi, chaque triplet supprimé ou ajouté est associé à un ou plusieurs types de changements définis dans cette ontologie. Plus précisément, dans cette dernière étape de représentation de changements, étant données les deux ensembles d'assertions ajoutées et supprimées et les deux versions old et new de la source de données, l'outil détecte les types sémantiques (classes de l'ontologie O^{DE}) à associer à chaque assertion ajoutée ou supprimée. A l'issue de cette étape, on obtient une ontologie O^{DE} peuplée avec les triplets réifiés avec les types de changements qu'ils représentent. On note que l'on peut associer plusieurs types (i.e. classes de l'ontologie O^{DE}) à un même triplet.

L'ontologie peuplée peut ensuite être exploitée pour répondre à des requêtes d'experts souhaitant obtenir des informations sur l'évolution d'une source de données. Il est également possible de répondre à des requêtes plus complexes, comme par exemple : la liste des nouvelles propriétés et les objets associés, la liste des changements concernant une URI donnée et la liste des propriétés fonctionnelles ayant subit des changements de valeurs.

Nous présentons en sous-section 2.1, l'ontologie O^{DE} puis la méthode de peuplement de cette dernière en sous-section 2.2.

2.1 Ontologie d'évolution (O^{DE})

O^{DE} est une ontologie qui permet de représenter sémantiquement es évolutions qui peuvent surgir entre deux versions d'une source de données. En effet, moyennant une classification plus fine des types de changements au niveau des données, il est possible de représenter avec plus de précision les changements au niveau des données mais également de déduire des changements possibles au niveau du schéma (ontologie). Par ailleurs, avec cette représentation plus riche, il est possible pour un expert de domaine de poser des requêtes simples (e.g. le nombre

d'instances supprimées) ou plus complexes (e.g. les propriétés fonctionnelles dont la valeur a été modifiée). Enfin, à travers l'ontologie O^{DE}, il est possible de fournir aux outils dédiés aux tâches d'intégration de données l'ensemble des triplets impliqués dans des changements qui sont susceptibles d'impacter leur résultats. Cela permettra à ces outils de recalculer leurs résultats uniquement sur la partie des données qui a été mise-à-jour.

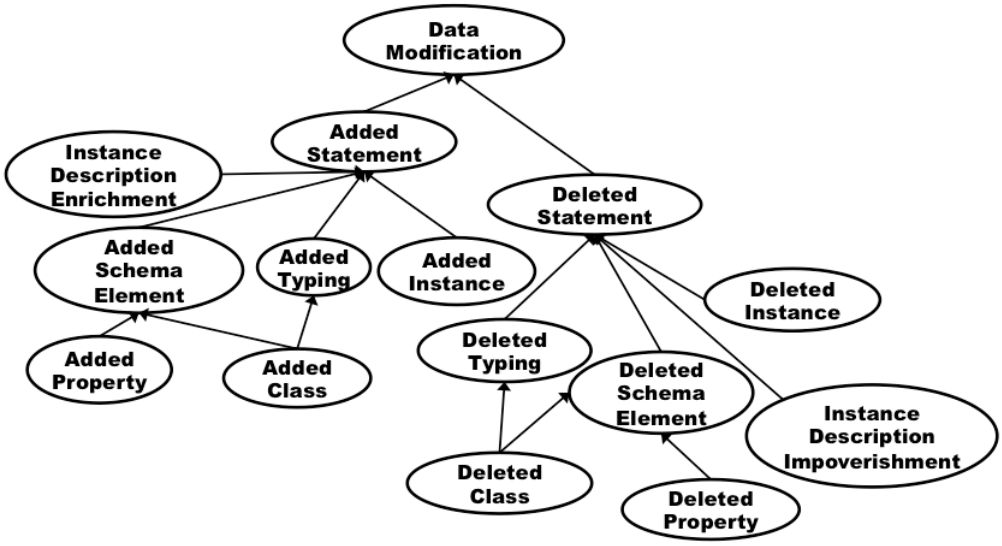

FIGURE 2 – Ontologie de changements dans les données RDF O^{DE}

Dans l'ontologie O^{DE} montrée en Figure 2, deux types généraux de changements sont distingués : *AddedStatement* et *DeletedStatement*. Pour chacun de ces types quatre sous-types sont associés. Pour le type *AddedStatement* on décrit :

- *AddedSchemaElement* qui représente les assertions impliqués dans l'introduction d'une nouvelle propriété (voir sous-type *AddedProperty*) ou d'une nouvelle classe (voir sous-type *AddedClass*) dans la nouvelle version de la source.
- *AddedTyping* qui décrit les assertions concernant l'introduction de nouveaux types pour des instances existantes (i.e. tous les nouveaux rdf :type). Parmi, ces assertions, certaines concernent la première utilisation d'une classe dans la source de données, d'où le lien de généralisation entre *AddedClass* et *AddedTyping* dans l'ontologie.
- *AddedInstance* qui représente l'ensemble des assertions décrivant des instances nouvelles dans la source de données.
- *InstanceDescriptionEnrichment* qui décrit l'ensemble des assertions qui viennent enrichir des instances existantes (assertions de la forme $<s\ p\ o>$ avec s existant dans l'ancienne source ou o existant dans l'ancienne source).

On note ici qu'une propriété *hasAddedInstance* de type *owl :ObjectProperty* est ajoutée à la classe *AddedInstance* pour garder trace des instances ayant été nouvellement introduites dans

la version new courante de la source (la classe étant représentée par un ensemble de triplets, il faut garder trace de l'URI de l'instance nouvelle sachant qu'il peut s'agir du sujet ou de l'objet des triplets). Par ailleurs, nous précisons que nous ne détaillerons pas la partie *DeletedStatement* de l'ontologie puisque celle-ci peut être décrite de façon strictement symétrique à celle de *AddedStatement*.

Les instances de l'ontologie O^{DE} sont des triplets réifiés par l'ajout d'un identifiant aux triplets. Ainsi, pour tout triplet <s, p, o> apparaissant dans l'ensemble de triplets ajoutés ou dans l'ensemble de triplets supprimés, on obtient la représentation réifiée suivante :

```
<tripleID-1 rdf:type rdf:Statement > .
<tripleID-1 rdf:subject s> .
<tripleID-1 rdf:predicate p> .
<tripleID-1 rdf:object o > .
<tripleID-1 rdf:type ClasseODE> .
```

2.2 Méthode de peuplement de O^{DE}

Après avoir détecté les changements élémentaires, en terme d'ajouts et de suppression d'assertions, entre deux versions old et new d'une source de données S, nous nous intéressons à la représentation sémantique des changements. Pour ce faire, nous avons développé deux algorithmes PODEA et PODES qui permettent de peupler l'ontologie O^{DE} en exploitant à la fois le fichier d'ajouts (resp. de suppressions) et le contenu de la version new (resp. old) de la source S.

Dans la suite nous ne présentons que l'algorithme PODEA (voir algorithme 1), puisque la description de l'algorithme PODES est analogue à celle de PODEA.

Dans l'algorithme PODEA, la fonction $existeA(s, p, o)$ vérifie l'existence du triplet (s, p, o) dans la version TDB_{old} de la source S. Une fonction $existeS(s, p, o)$ analogue est utilisée dans l'algorithme PODES pour vérifier l'existence du triplet (s, p, o) dans la version TDB_{new} de la source S

La fonction $instancier(c, tr)$ permet d'instancier la classe c de O^{DE} par le triplet réifié tr.

3 Interrogation de l'ontologie d'évolution (O^{DE})

L'ontologie O^{DE} peuplée par les triplets réifiés impliqués dans les changements peut être interrogée par un expert de domaine par des requêtes simples ou complexes représentées en SPARQL. Les requêtes simples que l'on peut exécuter permettent d'obtenir les instances O^{DE}. Ainsi, un expert de domaine pourra analyser les triplets qui induisent certains types de changements au niveau conceptuel ou au niveau des données. Par exemple, il pourra être intéressé par l'ensemble des classes qui ne sont plus instanciées dans la source de données ou par la liste complète des instances dont les triplets supprimés ont conduit à ce qu'une classe ne soit plus instanciée (voir table 1) .

L'expert pourra également poser des requêtes plus complexes combinant différents types d'informations : différentes classes de l'ontologie O^{DE}, des connaissances du domaine (e.g. fonctionnalité des propriétés) et des triplets des deux versions de la source de données considérée. La requête présentée en table 2 permet par exemple de retrouver la liste des instances de la propriété fonctionnelle $adressePrincipale$ dont la valeur a été modifiée (i.e. le triplet

Algorithme 1: PODEA – Peuplement de O^{DE} avec des changements de type ajouts –

Input :
– T_a : l'ensemble de triplets RDF ajoutés
– TDB_{old} : la version old de la source de données
– O^{DE} : l'ontologie d'évolution de données
Output : O^{DE}, l'ontologie d'évolution peuplée avec les changements T_a survenus dans TDB_{old}

```
for each (triplet t(subject, predicate, object) ∈ Ta) do
    if (predicate == rdf :type) then
        if (¬ existA( ?s, predicate, object)) then
            instantiate(AddedClass, reification(t))
        else
            instantiate(AddedTyping, reification(t))

    else
        if (¬ isLiteral(object) and (¬ (existA( ?s, ?p, object) or existA(object, ?p, ?o)))) then
            instantiate(AddedInstance, reification(t))
            instantiateHasAddedInstance(t, object)
        else if (¬ isLiteral(object)) then
            instantiate(InstanceDescriptionEnrichment, reification(t))
    if (¬ (existA( ?s, ?p, subject) or existA(subject, ?p, ?o))) then
        instantiate(AddedInstance, reification(t))
        instantiateHasAddedInstance(t, subject)
    else
        instantiate(InstanceDescriptionEnrichment, reification(t))
    if (¬ existA( ?s, predicate, ?o)) then
        instantiate(AddedPoperty, reification(t))
```

TABLE 1 – Requêtes simples exploitant la classe DeletedClass

/* Liste des classes supprimées

```
SELECT DISTINCT ?deletedClass
WHERE {
?node rdf:type   ode:DeletedClass
?node rdf:object ?deletedClass .
}
```

/* Liste des instances ayant conduit à une suppression de classe

```
SELECT ?s
WHERE {
?node rdf:type ode:DeletedClass .
?node rdf:subject ?s .
}
```

ayant même sujet pour le prédicat $adressePrincipale$ dans $ode : AddedStatement$ et dans $ode : DeletedStatement$).

Des requêtes SPARQL peuvent aussi être définies pour construire des fichiers de données représentant les différents types de changements pouvant avoir un impact sur les résultats d'un outil réalisant une tâche liées à l'intégration de données. Par exemple, pour le liage de données on pourrait définir une série de requêtes SPARQL dont les résultats permettraient de mettre à jour le résultat du liage en considérant la partie des données mises-à-jour susceptible d'impacter

TABLE 2 – Requête listant les modifications des valeurs d'une propriété fonctionnelle

```
SELECT ?subject  ?valueBefore ?valueAfter
WHERE {
?node      rdf:type      ode:AddedStatement .
?node      rdf:subject   ?subject .
?node      rdf:predicate <http://.../adressePrincipale> .
?node      rdf:object    ?valueAfter .
?othernode rdf:type      ode:DeletedStatement .
?othernode rdf:subject   ?subject .
?othernode rdf:predicate <http://.../adressePrincipale> .
?othernode rdf:object    ?valueBefore .
}
```

les résultats. La plupart des approche de liage (Saïs *et al.* (2009); Volz *et al.* (2009); Nikolov *et al.* (2012)) de données s'appuient sur des ensemble de propriétés discriminantes (Symeonidou *et al.* (2014)) pour détecter des liens d'identité entre données. Les approches peuvent être très couteuses. Ainsi, en utilisant cette approche il est possible de fournir les nouvelles instances et les modifications qu'ont subit les propriétés discriminantes pour les instances existantes.

4 Experimentations

L'objectif de ces expérimentations est de montrer que l'approche proposée permet de peupler l'ontologie des modifications O_{DE} lorsque qu'une source de données subit un nombre important de changements. Il s'agit également de montrer comment l'ontologie, une fois peuplée, peut être utilisée pour étudier les différents types de changements.

4.1 Description des données

Nous avons évalué notre approche en utilisant les versions 3.5, 3.8 et 3.9 de DBPedia [1]. Nous nous sommes intéressés aux données décrites dans la classe $Person$ (fichier PersonData) auxquelles ont été ajoutées toutes les données de typage concernant ces personnes. Le tableau 3 présente le nombre de triplets, le nombre d'instances de personnes, le nombre de propriétés ainsi que le nombre de types différents associés à ces personnes dans ces trois différentes versions.

TABLE 3 – Evolution des triplets décrivant les Personnes dans trois versions de DBPedia

	Version 3.5	Version 3.8	Version 3.9
#triplets	482 080	18 719 429	22 008 122
#instances	48 692	2 853 529	3 733 629
#propriétés	9	9	9
#types	71	348	434

1. http ://dbpedia.org/services-resources/datasets

Entre la version 3.5 et la version 3.9, le nombre de triplets de la classe $Person$ a été multiplié par 45, le nombre de classes typant ces instances a été multiplié par 6 mais le nombre de propriétés n'a pas été modifié. Le nombre de changements étant important, cela nous a semblé un bon exemple pour tester notre approche.

4.2 Résultats et discussion

Nous avons détecté les changements élémentaires entre deux versions successives de la classe $Person$ et obtenu les deux fichiers contenant pour l'un, les triplets ajoutés et pour l'autre les triplets supprimés (temps d'exécution inférieur à 10 mn). Ces deux fichiers ont été exploités pour peupler l'ontologie O_{DE} (temps d'exécution sur plus de 18 millions de triplets : 55 mn). Lorsque les versions v3.5 et v3.8 sont comparées, après peuplement de l'ontologie O_{DE}, plus de 18 millions d'assertions sont instances de la classe $DataModification$. Il faut bien sûr noter que leur réification entraîne un sur-coût en terme de représentation : ces 18 millions d'assertions sont représentées par plus de 155 millions de triplets.
La taille des données représentant les personnes ayant été multipliée par 39, presque toutes ces assertions sont de type $AddedStatement$ (tab. 4), mais près de la moitié des triplets existants ont été supprimés (classe $DeletedStatement$, tab.5). La table 4 (resp. 5) montre comment les assertions instances de la classe $AddedStatement$ (resp. $deletedstatement$) se répartissent dans les différentes classes de l'ontologie O_{DE}. Le temps d'exécution est également donné pour chaque type de changement.

TABLE 4 – Type et nombre de changements pour la classe $AddedStatement$

	#Added Statements	#Added SchemaElement	#Added Property	#Added Class	#Added Typing	#Added Instance
v3.5 -> v3.8	18 469 394 (12 :43 mn)	284 (5 :16 mn)	5 (3 :28 mn)	279 (1 :48 mn)	13 596 447 (6 :43 mn)	2 835 666 (7 :20 mn)
v3.8 ->v3.9	4 813 958 (01 :49 mn)	86 (14 s)	0 (2 s)	86 (12 s)	4 015 870 (1 :21 mn)	1 103 520 (45 s)

TABLE 5 – Type et nombre de changements pour la classe $DeletedStatement$

	#Deleted Statements	#Deleted SchemaElement	#Deleted Property	#Deleted Class	#Deleted Typing	#Deleted Instance
v3.5 -> v3.8	232 058 (12 :43 mn)	7 (29 s)	5 (9.5 s)	2 (3.5 s)	41 788 (6 s)	6 732 (8.6 s)
v3.8 ->v3.9	1 525 420 (35 s)	0 (3 s)	0 (2 s)	0 (2 s)	1 359 525 (23 s)	223 489 (25 s)

Les classes $DeletedInstance$ et $AddedInstance$ permettent d'observer comment les instances évoluent sans se limiter à l'évolution globale du nombre d'instances. Ainsi, si près de trois millions d'instances de $Person$ ont été ajoutées dans v3.8, 14% des instances de v3.5 ont été supprimées. De plus les classes $addedTyping$ et $deletedtyping$ montre que le typage des instances évolue (18% des triplets supprimés sont des typages d'instances). Une requête utilisant les instances supprimées/ajoutées et les types supprimées/ajoutées, montre que de nombreux typages ont évolués pour des instances qui sont présentes dans les deux versions

de la source (donc les ajouts ou suppressions de types ne pas uniquement dus à l'ajout ou suppression d'instances).

L'utilisation de cette approche pour étudier l'évolution des données permet également de détecter que les éléments de l'ontologie utilisés pour décrire ces personnes a évolué. Ainsi, entre v3.5 et v3.8, 284 classes sont apparues dans la description des personnes et deux ont disparu. Notre étude étant extensionnelle, les classes nouvelles correspondent soit à des classes qui ont été ajoutées dans l'ontologie, soit à des classes existantes dont l'extension ne comprenait pas d'instances de $Person$. De plus, les résultats montrent que cinq propriétés sont nouvellement utilisées pour décrire les personnes tandis que cinq autres ont disparu. En fait, les résultats des requêtes correspondantes montrent qu'il s'agit des mêmes propriétés dont l'URI a été modifiée (e.g. la propriété <http ://.../birth est devenu <http ://.../birthDate à partir de la version V5.8). Si l'on compare les deux versions suivantes, les propriétés n'ont pas évoluées. En revanche, 86 classes ont été ajoutées.

Une fois l'ontologie peuplée par les différentes assertions, celle-ci peut être utilisée pour exécuter d'autres requêtes qui se basent sur les classes définies dans O_{DE} . Ainsi, il est facile pour un expert d'étudier l'évolution de la description d'une resource en effectuant des requêtes sur la base de connaissances O_{DE}. Par exemple, nous avons utilisé une requête SPARQL pour rechercher les assertions de type $InstanceDescriptionEnrichment$ qui ont été ajoutées pour l'URI correspondant à Barack Obama (cf table 6). Les résultats ont montré que 7 assertions ont été ajoutées pour cette URI, telle que la propriété $description$ dont la valeur est dans v3.8 "American politician, 44th President of the United States"@en. De plus, en recherchant les assertions de la classe $AddedTyping$ le concernant, nous pouvons voir que cette URI est nouvellement typée par les classes $Agent$ et $OfficeHolder$.

TABLE 6 – Requête listant les nouveaux triplets enrichissant la description de B. Obama

```
SELECT ?property ?value
WHERE{
?node rdf:subject barrack_obama .
?node rdf:type ode::instanceDescriptionEnrichment
.
?node rdf:predicate ?property .
?node rdf:object ?value
.
```

D'autres requêtes ont été exécutées permettant par exemple de lister ou de compter les nouvelles instances d'une sous-classe de personne donnée (utilisation de la classe $addTyping$), ou de lister les instances qui ont été supprimées (utilisation de la classe $deleteTyping$). En executant la requête ci-dessous (voir table 7), nous avons pu voir que 57 595 artistes ont été ajoutés à la version 3.5.

Voici un extrait de la liste des artistes obtenus :
(?instance = <http ://dbpedia.org/resource/Bill$_Reid >)
($?instance =< http://dbpedia.org/resource/John\_White\_(colonist_and_artist) >$)

$(?instance =< http://dbpedia.org/resource/Louis\_Aragon >)$
$(?instance =< http://dbpedia.org/resource/JeanPierre\_Rampal >)$
$(?instance =< http://dbpedia.org/resource/Pierre\_de\_Marivaux >)$
$(?instance =< http://dbpedia.org/resource/Jean\_Giraudoux >)$
$(?instance =< http://dbpedia.org/resource/Eugene\_Ionesco >)$
$(?instance =< http://dbpedia.org/resource/Eric\_Dolphy >)$
$(?instance =< http://dbpedia.org/resource/Jacob\_Riis >)$
$(?instance =< http://dbpedia.org/resource/Baby\_Gramps >)$
$(?instance =< http://dbpedia.org/resource/Mark\_Mothersbaugh >)$
$(?instance =< http://dbpedia.org/resource/Frank\_Miller_{(}comics) >)$
$(?instance =< http://dbpedia.org/resource/Rosa\_Bonheur >)$
$(?instance =< http://dbpedia.org/resource/Sandro\_Botticelli >)$
$(?instance =< http://dbpedia.org/resource/Thomas\_Lawrence >)$
$(?instance =< http://dbpedia.org/resource/Geoffrey\_A.\_Landis >)$
$(?instance =< http://dbpedia.org/resource/Princess\_(singer) >)$
$(?instance =< http://dbpedia.org/resource/James\_Abbott\_McNeill\_Whistler >)$

TABLE 7 – Requête listant les instances qui ont acquis le type Artiste

```
SELECT ?instance
WHERE{
?node rdf:subject ?instance .
?node rdf:type rdfEvolution:addedTyping .
?node rdf:predicate rdf:type .
?node rdf:object <http://dbpedia.org/ontology/Artist> .
}
```

5 Conclusion

Dans cet article, nous avons présenté une approche originale de détection et de représentation sémantique des changements dans une source RDF. Plus précisément, nous avons conçu une ontologie O^{DE} qui représente et structure les différentes types de changements au niveau des données. A travers des requêtes SPARQL, il est possible pour un expert de données ou à une application réalisant une tâche d'intégration d'interroger l'ontologie peuplée par les triplets ajoutés ou supprimés.
Nous avons mené une expérimentation sur deux versions consécutives de DBPedia ainsi que sur une version ancienne et une versions plus récente. Cette expérimentation a montré que cette approche permet d'étudier l'évolution des données sur différents aspects. Les requêtes simples permettent de lister ou de compter les assertions instances de chacune des classes de O_{DE}. Des requêtes plus complexes, qui utilisent différentes classes de cette ontologie, peuvent être facilement définies pour s'adapter aux besoins d'un expert ou d'une application. De plus, malgré le processus de réification qui induit une représentation assez couteuse des changements, les temps d'execution des requêtes restent acceptables même pour une source qui a subit un nombre très volumineux de changements.

En perspectives, nous souhaitons étendre ce travail de façon à pouvoir détecter des changements dans des sources externes qui seraient liées à une source via des lien SameAs. Ainsi, un expert ou une application pourront être informée que des changements ont été réalisés concernant des resources décrites dans des sources de données externes mais pour lesquelles un lien d'identité a été déclaré (SameAs). Les changements au niveau d'une source pourront ainsi être propagés dans le Web de données en exploitant les liens SameAs entre les données. Nous envisageons, par ailleurs, d'étudier la possibilité d'inférer des changements au niveau des axiomes d'une ontologie,

par exemple la connaissance sur les propriétés fonctionnelles ou encore les cardinalité maximum et minimum de certaines propriétés. Enfin, nous souhaitons intégrer cette nouvelle approche de détection et de représentation des changements dans un processus d'intégration de données plus global où les tâches d'intégration exploiteront directement l'ontologie O^{DE} peuplée par notre approche pour réaliser des tâches d'intégration incrémentales.

Références

DINH D., DOS REIS J. C., PRUSKI C., SILVEIRA M. D. & REYNAUD-DELAÎTRE C. (2014). Identifying relevant concept attributes to support mapping maintenance under ontology evolution. *J. Web Sem.*, **29**, 53–66.

KÄFER T., UMBRICH J., HOGAN A. & POLLERES A. (2012). Dyldo : Towards a dynamic linked data observatory. In *WWW2012 Workshop on Linked Data on the Web, Lyon, France, 16 April, 2012*.

NIKOLOV A., D'AQUIN M. & MOTTA E. (2012). Unsupervised learning of link discovery configuration. In *ESWC*, p. 119–133.

OWENS A., SEABORNE A., GIBBINS N. & MC SCHRAEFEL (2008). Clustered tdb : A clustered triple store for jena.

PAPAVASILEIOU V., FLOURIS G., FUNDULAKI I., KOTZINOS D. & CHRISTOPHIDES V. (2013). High-level change detection in rdf(s) kbs. *ACM Trans. Database Syst.*, **38**(1), 1 :1–1 :42.

SAÏS F., PERNELLE N. & ROUSSET M.-C. (2009). Combining a logical and a numerical method for data reconciliation. *Journal on Data Semantics*, **12**, 66–94.

SYMEONIDOU D., ARMANT V., PERNELLE N. & SAÏS F. (2014). Sakey : Scalable almost key discovery in RDF data. In *The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part I*, p. 33–49.

VOLZ J., BIZER C., GAEDKE M. & KOBILAROV G. (2009). Discovering and maintaining links on the web of data. In *ISWC*, p. 650–665.

ZABLITH F., ANTONIOU G., D'AQUIN M., FLOURIS G., KONDYLAKIS H., MOTTA E., PLEXOUSAKIS D. & SABOU M. (2015). Ontology evolution : a process-centric survey. *Knowledge Eng. Review*, **30**(1), 45–75.