RDF data evolution: efficient detection and semantic representation of changes

Nathalie Pernelle
LRI, Université Paris Sud
92450 Orsay, France
Nathalie.Pernelle@lri.fr

Fatiha Saïs
LRI, Université Paris Sud
92450 Orsay, France
Fatiha.Sais@lri.fr

Daniel Mercier
Université Paris Sud
92450 Orsay, France
daniel.mercier@u-psud.fr

Sujeeban Thuraisamy
Université Paris Sud
92450 Orsay, France
sujeeban.thuraisamy@u-psud.fr

ABSTRACT

Many RDF data sources are constantly changing for both data and vocabulary (ontology) levels. Many integration tasks are impacted by these changes. In this context, it is important to develop approaches to detect and represent these changes. Many studies have focused on the detection, the representation and the management of changes at the ontology level. In this paper, we present an approach which allows to detect and represent elementary and complex changes that can be detected when we focus only on the data level. A first experiment was conducted on different versions of DBpedia.

CCS Concepts

•**Information systems** → **Information integration;**

Keywords

Data evolution; Data changes; RDF; Ontologies

1. INTRODUCTION

Today, we are experiencing an unprecedented production of resources, published as Linked Open Data (LOD, for short). This is leading to the creation of a global data space containing billions of RDF triples from different domains (e.g. DBpedia, Yago, GeoNames, FMA). One of the intrinsic features of the LOD is its dynamicity. LOD datasets are continuously evolving for different reasons: information enrichment and correction, scientific knowledge is constantly growing thanks to the progress of scientific research and every day big data production: *"Every day, we create 2.5 quintillion bytes of data [2]"*.

In the Web of data, when a dataset evolves, the changes may concern the ontological level where changes may involve classes, properties, axioms and mappings to other ontologies. The changes may also concern the instance level where data modifications may affect resources typings, property values, or identity links between resources.

SEMANTICS 2016: Posters and Demos Track
September 13-14, 2016, Leipzig, Germany

Many data integration tasks (e.g. synchronization, data linking or fusion) are directly impacted by the data evolution, that may lead to incomplete or incorrect results. Therefore, it is important to develop tools that allow to detect the changes and semantically represent them in a way to be comprehensible and interpretable by human experts and applications. This should be achieved in a manner that supports the development of incremental approaches for data integration tasks, that is, tools performing these tasks can update their results without considering the entire content of the updated datasets.

An extensive work has been conducted on detection, representation and management of the changes at the ontological level (see [5] for a survey). Our approach is more related to works that studied evolution of datasets at the data (instance) level. The existing approaches may be classified into two categories. First, some works [6, 1] only focus on low-level changes (i.e. addition and deletion). [6, 1] studied both ontology and data low-level changes. [6] proposed a new formalism for low-level change detection in RDFS datasets. [1] discusses low-level changes for propositional Knowledge Bases by providing formal properties as delta uniqueness and change reversibility. Some recent approaches have investigated more high-level changes. [4] have proposed the notion of simple and complex changes in RDF datasets. The complex changes are represented in an ontology of changes and allow to provide a more synthetic and comprehensible representation of the changes. However, they do not declare direct links between the RDF triples and the induced changes, which do not allow to perform specific queries on the modified triples. [3] proposed a fixed and predefined set of abstract changes without giving abilities to answer queries that combine different kinds of changes.

In this paper, we propose an approach that detects and semantically represents high-level changes at the instance level of RDF datasets. We have designed an ontology O^{DE} in which semantic types of the changes are formally defined. Once the delta (i.e. the set of added and deleted triples) is computed, we apply two generic algorithms PODEA (Population of O^{DE} with Addition changes) and PODES (Population of O^{DE} with Suppression changes) that allow to populate the O^{DE} ontology by reifying the added/deleted RDF triples. The populated ontology O^{DE} may be exploited to perform simple queries that concern one class and more complex queries that may involve several classes of O^{DE} and additional knowledge.

In section 2, we present our approach of detection and semantic representation of changes in RDF dataset. We then show in 3, how the populated ontology may be queried using SPARQL queries. After that, we present in section 4 the results of our experimental evaluation on three different versions of DBpedia. Finally, we conclude and give some future plans in section 5.

2. DETECTION AND SEMANTIC REPRESENTATION OF CHANGES IN RDF DATA

In this section, we present first some preliminaries and the overview of our approach. Then, we describe the ontology O$_{DE}$. Finally, we will describe the algorithm that allows to populate the O$_{DE}$ ontology.

2.1 Preliminaries

We consider RDF datasets as a set of RDF triples conforming to an OWL ontology.
An RDF triple is in the form $< i \;\; p \;\; v >$ where: i is a URI, p a property and v a URI or a literal.
When an ontology O is available, it can be defined as (C, P, A) where: C is the set of classes, P is the set of datatype or object properties, and A is the set of axioms including subsumption relation between classes and properties.

2.2 Approach overview

Let consider two different versions D_{old} and D_{new} of an RDF dataset[1], the approach first computes the set $diff = T_d \cup T_a$ with $T_d = D_{old}/D_{new}$ and $T_a = D_{new}/D_{old}$. Then, an analysis step is applied to determine the type(s) of each RDF triple that belongs to $diff$. These typed triples are then used to populate the data evolution ontology O$_{DE}$. Thus, the obtained representation can be exploited using SPARQL queries by tools or users to answer their specific needs.

2.3 Data evolution ontology O$_{DE}$

Instead of representing the evolution of an RDF dataset only in terms of added and deleted triples, we have defined the O^{DE} ontology to formalize different kinds of changes (see Figure 1).

Two general types of changes are distinguished: the classes *AddedStatement* and *DeletedStatement*. The subclasses of *AddedStatement* are the following:
AddedTyping, represents the triples that specify a new type for a given instance i. More formally,
$\forall t \in T_a$, $AddedTyping(t, c, i)$ iff $t =< i \;\; \textit{rdf:type} \;\; c >$
AddedSchemaElement, represents the triples that involve a new property[2] in the dataset (sub-class *AddedProperty*) or a new class (sub-class *AddedClass*) in D_{new}. More formally,
$\forall t \in T_a$, $AddedClass(t, c)$ iff $t =< i \;\; \textit{rdf:type} \; c >$
and $\nexists \, t' \in D_{old}$ s.t. $t' =< i' \;\; \textit{rdf:type} \; c >$ and $i \neq i'$

[1]If the ontology exists, datasets are saturated
[2]new, means that the property was not instantianted in D_{old}. This does not mean that the property was not declared in the ontology.

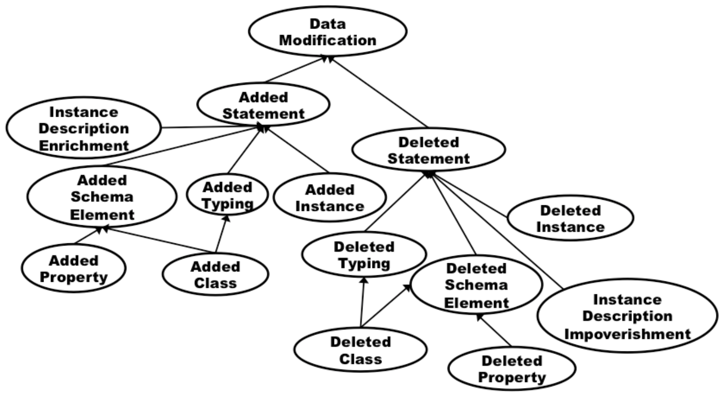

Figure 1: RDF data evolution Ontology O^{DE}

$\forall t \in T_a$, $AddedProperty(t, p)$ iff
$t =< i \; p \; v >$ and $\nexists \, t' \in D_{old}$ s.t. $t' =< i' \; p \; v' >$ and $i \neq i'$ and p is a property
Note that all the triples typed as *AddedClass* are also typed as *AddedTyping*.
AddedInstance represents the triples that involve a new instance i. More formally,
$\forall t \in T_a$, $AddedInstance(t, i)$ iff
$\nexists \, t' \in D_{old}$ s.t. $t' =< i \; p \; v >$ or $t' =< i' \; p \; i >$
InstanceDescriptionEnrichment represents the triples that involve a new property instance for an existing instance i.
$\forall t \in T_a$, $InstanceDescriptionEnrichment(t, i)$ iff
$t =< i \;\; p \;\; v >$ and $\exists \; t' \in D_{old}$ s.t. $t' =< i \; \_ \; \_ >$ or $t' =< \_ \, \_ \, i >$.

We will not detail the *DeletedStatement* subclasses, since their definition is analogous. It suffices to replace in the definition D_{old} by D_{new} and T_a by T_d.

The instances of O^{DE} classes correspond to reified RDF triples. For each added or deleted triple, a property *rdf:type* is added for each O^{DE} class that types this triple. We note that, when a triple corresponds to several types of changes, its corresponding reification ID is used represent the different changes (see Example 1).

Example 1. Let t =<dbp:Louis_Aragon dbp:birthDate "1897"> be an added triple typed as *AddedProperty* and *InstanceDescriptionEnrichment*. Its reified representation is as follows:

```
<tripleID-1 rdf:type rdf:Statement > .
<tripleID-1 rdf:subject dbp:Louis_Aragon> .
<tripleID-1 rdf:predicate  dbp:birthDate> .
<tripleID-1 rdf:object "1897" > .
<tripleID-1 rdf:type ode:AddedProperty> .
<tripleID-1 rdf:type ode:InstanceDescriptionEnrichment> .
```

2.4 O^{DE} population algorithm

We have developed two algorithms PODEA and PODES that allow to detect and associate a semantic type for each added (suppressed, resp.) triple to D_{old} (from D_{new}, resp.). These triples are reified and used to instantiate the O^{DE} ontology, described above. We only present PODEA algorithm (see algorithm 1) that allows to populate the O^{DE} ontology by exploiting the set of added triples. Indeed, PODES description is analogous to PODEA. In PODEA algorithm, the function $existA(s, p, o)$ checks the existence of the triple (s, p, o) in D_{old}. The function $instantiate(c, tr)$ allows to instantiate the class c of O^{DE} by the reified triple t_r. Finally, the function $instantiateHasAddedInstance(t, URI)$ allows to keep track of the URI of the added instance (subject or object).

Algorithm 1: PODEA – O^{DE} population with added triples –

```
Input:
  • $T_a$: set of added RDF triples.
  • $D_{old}$: the old version of the dataset
  • $O^{DE}$: data evolution ontology
Output: $O^{DE}$, data evolution ontology populated by the
        changes $T_a$ which arose in $D_{old}$
for each (triplet t(subject, predicate, object) $\in T_a$) do
    if (predicate == rdf:type) then
        if ($\neg$ existA(?s, predicate, object)) then
            instantiate(AddedClass, reification(t))
        else
            instantiate(AddedTyping, reification(t))
    else
        if ($\neg$ isLiteral(object) and ($\neg$ (existA(?s, ?p, object)
        or existA(object, ?p, ?o)))) then
            instantiate(AddedInstance, reification(t))
            instantiateHasAddedInstance(t, object)
        else if ($\neg$ isLiteral(object)) then
            instantiate(InstanceDescriptionEnrichment,
            reification(t))
    if ($\neg$ (existA(?s, ?p, subject) or existA(subject, ?p, ?o)))
    then
        instantiate(AddedInstance, reification(t))
        instantiateHasAddedInstance(t, subject)
    else
        instantiate(InstanceDescriptionEnrichment,
        reification(t))
    if ($\neg$ existA(?s, predicate, ?o)) then
        instantiate(AddedPoperty, reification(t))
```

3. O^{DE} QUERYING

The populated ontology O^{DE} can be queried by a domain expert or a data integration application using SPARQL queries. Thus, a domain expert may analyze the triples which induce some kinds of changes at the data level. Simple queries allow to obtain instance descriptions of one O^{DE} ontology class. For example, a simple query may be constructed to analyse the triples that have enriched the description of Barack Obama (see Q2 in Table 1). Some of these simple data changes may show that some ontology elements are not instantiated anymore. For example, an expert may be interested by the set of classes that are not any more instantiated in the dataset (see Q1 in Table 1).

More complex queries can combine different kinds of information: different O^{DE} classes, domain knowledge (e.g. functionality of some properties) and triples of two versions of the considered dataset. The query presented in Table 2 allows, for instance, to retrieve the set of instances of the functional property *mainAddress* for which the value has been modified.

Furthermore, SPARQL queries can also be defined to obtain a set of triples that may have an impact on the results of a data integration task. For example, for data linking task, that aims at detecting identity links (i.e. *owl:sameAs*) between different descriptions that refer to the same world entity, we may define a set of SPARQL queries to obtain the data changes that may affect the data linking results (i.e. to find new links or erroneous links).

Table 1: Simple queries.

```
//Q1:List of classes that no longer have instances

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

```
//Q2:List of the added triples that enrich the description
of B. Obama

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

Table 2: A query that retrieve modified values for a functional property.

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

4. FIRST EXPERIMENTS

The main objective of this first evaluation is to show that the proposed approach allows to populate the data evolution ontology O_{DE} when a data source incurs an important number of changes. Furthermore, through this experiment, we aimed to show how O_{DE}, when populated, may be used to answer to simple expert queries but also to more complex ones.

Table 3: Evolution of RDF triples describing Persons in the three versions 3.5, 3.8 et 3.9 of DBPedia

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

Our approach has been implemented using Java programming language and TDB–Jena Triples Database is used to access and store the RDF triples.

We have evaluated our approach by using three versions of DBPedia[3] (versions v3.5, v3.8 and v3.9). We focused our study on the instances of the class *Person* (PersonData file). We have added to this file all the typing information of these persons, i.e. all the *rdf:type*) triples. Table 3 presents the

[3]http://dbpedia.org/services-resources/datasets

Table 4: Type and number of changes for the class *AddedStatement*

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

number of RDF triples, the number of instances and the number of associated types of instances of the class $Person$ in the considered three versions.

Between the version v3.5 and the version v3.9, the number of triples of the class $Person$ has been multiplied by 45, the number of classes that type these instances has been multiplied by 6.

The experiments have been executed on a single machine with 8GB RAM, processor core i7-3517U 1.90GHz. We have, first, applied our approach on successive versions of the class $Person$ and detected low-level changes: added triples and deleted triples (execution time lower than 10 min). Then these two files have been exploited to populate O_{DE} ontology. When the two versions v3.5 et v3.8 are compared, more than 18 millions of assertions are associated to the class $DataModification$ (execution time : 35 min). It should be noted that, due to the reification step, these 18 millions of assertions are represented by more than 155 millions of triples.
Almost all the assertions are of type *AddedStatement* (see Table 4). However, around half of existing triples have been deleted (class *DeletedStatement*). Table 4 show how instances of the class *AddedStatement* have been classified in O_{DE}. Execution time is also given for each type of change.

The classes *DeletedInstance* and *AddedInstance* allow to observe how the data evolve. Thereby, when around three millions of $Person$ class instances have been added in v3.8, 14% of instances have been deleted in v3.5. Furthermore, the classes *AddedTyping* and *DeletedTyping* show that the typing of the instances evolve (18% of the deleted triples are triples types). A complex query using the Deleted/AddedInstance and the Deleted/AddedTyping classes, shows that a big number of types have evolved for instances that are described in the two versions of DBPedia (i.e. addition and deletion of types is not only due to the addition and the deletion of instances).
We have also used O_{DE} to detect if the ontology elements that are used to describe these persons have evolved. Thus, between the versions v3.5 and v3.8, 284 classes have appeared in the description of the persons and two are not used anymore. Furthermore, the results show that five properties have been used for the first time to describe the persons while five others are lost. In fact, it corresponds to properties for which the URI has been modified (e.g. the property<http://.../birth> became <http://.../birthDate> in v3.8 version).

Once the ontology is populated, this one may be used to perform other simple queries based on the classes defined in O_{DE}. For example, we have used a SPARQL query to retrieve assertions of type *InstanceDescriptionEnrichment* that are added for the URI corresponding to *Barack Obama* (see Q2 in Table 1). The results showed that 7 assertions have been added to this URI (e.g. a property *description* which has as value in v3.8 "*American politician, 44th President of the United States*"@en). In addition to this, by looking for assertions of the class *AddedTyping* that concern the URI of *Barack Obama*, we can see that this URI has been associated to new types (*Agent* and $OfficeHolder$).

By executing a simple query, a user can also detect that 57 595 artists have been added to the version 3.5.

5. CONCLUSION

In this paper, we have presented an approach that detects and semantically represents data changes in RDF datasets. To represent these changes, we have designed an ontology named O^{DE} that is automatically populated using our approach. In O^{DE}, a class instance is a reified triple that belongs to the set of added or deleted triples computed between successive versions of one dataset. This populated ontology can then be queried using SPARQL to analyse various simple or complex changes. Our experiments have shown that our approach can be applied to large datasets in which many changes have occurred.

6. REFERENCES

[1] E. Franconi, T. Meyer, and I. Varzinczak. Semantic diff as the basis for knowledge base versioning. In *NMR*, 2010.

[2] A. Gandomi and M. Haider. Beyond the hype: Big data concepts, methods, and analytics. *International Journal of Information Management*, 35(2):137 – 144, 2015.

[3] V. Papavasileiou, G. Flouris, I. Fundulaki, D. Kotzinos, and V. Christophides. High-level change detection in rdf(s) kbs. *ACM Trans. Database Syst.*, 38(1):1:1–1:42, Apr. 2013.

[4] Y. Roussakis, I. Chrysakis, K. Stefanidis, G. Flouris, and Y. Stavrakas. A flexible framework for understanding the dynamics of evolving RDF datasets. In *The Semantic Web - ISWC 2015 - Bethlehem, PA, USA, October 11-15, 2015*, pages 495–512, 2015.

[5] F. Zablith, G. Antoniou, M. d'Aquin, G. Flouris, H. Kondylakis, E. Motta, D. Plexousakis, and M. Sabou. Ontology evolution: a process-centric survey. *Knowledge Eng. Review*, 30(1):45–75, 2015.

[6] D. Zeginis, Y. Tzitzikas, and V. Christophides. On computing deltas of rdf/s knowledge bases. *ACM Trans. Web*, 5(3):14:1–14:36, July 2011.