Specification and Incremental Maintenance of Linked Data Mashup Views

Vânia M.P. Vidal[1], Marco A. Casanova[2(✉)], Narciso Arruda[1], Mariano Roberval[1], Luiz Paes Leme[3], Giseli Rabello Lopes[4], and Chiara Renso[5]

[1] Department of Computing, Federal University of Ceará, Fortaleza, CE, Brazil
{vvidal,narciso,mariano}@lia.ufc.br
[2] Department of Informatics,
Pontifical Catholic University of Rio de Janeiro, Rio de Janeiro, RJ, Brazil
casanova@inf.puc-rio.br
[3] Fluminense Federal University, Niteroi, RJ, Brazil
lapaesleme@ic.uff.br
[4] Computer Science Department,
Federal University of Rio de Janeiro, Rio de Janeiro, RJ, Brazil
giseli@dcc.ufrj.br
[5] ISTI Institute of National Research Council, Pisa, Italy
chiara.renso@isti.cnr.it

Abstract. The Linked Data initiative promotes the publication of previously isolated databases as interlinked RDF datasets, thereby creating a global scale data space, known as the Web of Data. Linked Data Mashup applications, which consume data from the multiple Linked Data sources in the Web of Data, are confronted with the challenge of obtaining a homogenized view of this global data space, called a Linked Data Mashup view. This paper proposes an ontology-based framework for formally specifying Linked Data Mashup views, and a strategy for the incremental maintenance of such views, based on their specifications.

Keywords: Data mashup application · RDF dataset interlinking · Linked data · View maintenance

1 Introduction

The Linked Data initiative [2] brought new opportunities for building the next generation of Semantic Mashup applications [8]. By exposing previously isolated datasets as data graphs, which can be interlinked and integrated with other datasets, Linked Data allows creating a global-scale interlinked data space, known as the Web of Data. The success of the Linked Data initiative is mainly due to the adoption of known Web standards, such as Web infrastructure standards (URIs and HTTP), Semantic Web standards (RDF and RDFS) and vocabularies, which facilitate the deployment of Linked Data sources.

A *linked data mashup* is an (Web) application that offers new functionality by combining, aggregating, and transforming data available on the *Web of Data* [11], [19]. Thanks to the new technologies that have been developed by the Semantic Web community, a great amount of linked data mashups were recently produced [11], in several domains. A simple example of Linked Data Mashup is *BBC Music* [13] which integrates data from two Linked Data sources, *DBpedia* [3] and *MusicBrainz* [20].

Linked Data Mashup (LDM) applications are confronted with the challenge of obtaining a homogenized view of this global data space, which we call a *Linked Data Mashup view* (*LDM view*). The creation of a LDM view is a complex task which involves four major challenges: (1) selection of the Linked Data sources that are relevant for the application; (2) extraction and translation of data from different, possibly heterogeneous Linked Data sources to a common vocabulary; (3) identification of links between resources in different Linked Data sources; (4) combination and fusion of multiple representations of the same real-world object into a single representation and resolution of data inconsistencies to improve the quality of the data.

To be useful, a LDM view must be continuously maintained to reflect dynamic data sources updates. Basically, there are two strategies for materialized view maintenance. *Re-materialization* re-computes view data at pre-established times, whereas *incremental maintenance* periodically modifies part of the view data to reflect updates to the database. It has been shown that incremental maintenance generally outperforms full view recomputation [1,7,9,15].

In this paper, we investigate the problem of incremental maintenance of LDM views. First, we propose an ontology-based framework for formally specifying LDM views. In our framework, an LDM view is specified with the help of exported views, sameAs linkset views, data fusion rules and a normalization function. The LDM view specification is used to automatically materialize the mashup view. Then, we propose a strategy that uses the LDM view specification for incrementally maintain the mashup view materialization.

The incremental maintenance strategy of LDM views is the major contribution of this paper. The strategy addresses the problem of dealing with the combination and fusion of multiple representations of the same real-world object when the Linked Data sources are updated. As discussed in the related work section, this problem received little attention and yet poses new challenges due to very nature of mashup data.

The paper is organized as follows. Section 2 presents the framework to create LDM views. Section 3 discusses how to maintain LDM Views. Section 4 reviews related work. Finally, Section 5 contains the conclusions.

2 Ontology-Based Framework for LDM View Specification

2.1 Overview

In this section, we discuss a three level ontology-based framework, as summarized in Figure 1, to formally specify LDM views. In the Mashup View Layer, the mashup view ontology O_D specifies the concepts of the mashup application (i.e., the *conceptual model*), which is the common vocabulary for integrating data exported by the Linked Data sources.

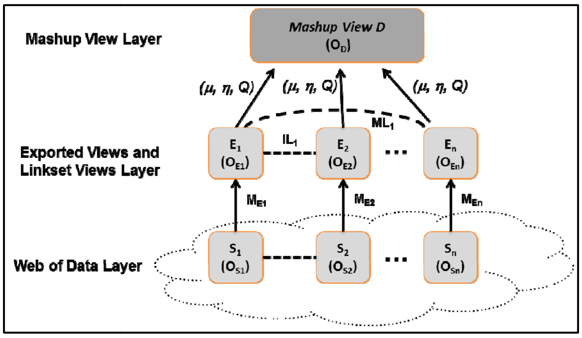

Fig. 1. Three Level Ontology-Based Framework

In the Web of Data Layer, each data source S_i is described by a *source ontology* O_{Si}, published on the Web according to the Linked Data principles. These source ontologies are depicted in the Web of Data layer in Figure 1.

Each Linked Data source S_i exports one or more views. Each such view E_i has an ontology O_{Ei} and a set of rules M_{Ei} that map concepts of O_{Si} to concepts of O_D. The vocabulary of O_{Ei} is the subset of the vocabulary of O_D whose terms occur in the head of the rules in M_{Ei}. The exported ontologies are depicted in the Exported Views and Linkset Views Layer in Figure 1.

We consider two types of sameAs links: *exported sameAs links*, which are exported by a Linked Data source, and *mashup sameAs links*, which are automatically created based on a sameAs linkset view specification [5] specifically defined for the mashup application.

As detailed in the following subsections, a *LDM view specification* is an n-tuple $\lambda = (D, O_D, E_1, ..., E_n, EL_1,..., EL_p, ML_1,...,ML_q, \mu, \eta, Q)$, where:

- *D* is the *name* of the mashup view
- O_D is the *mashup view ontology*
- E_1, ..., E_n are *exported view specifications* with ontologies O_{E1},...,O_{En}, whose vocabularies are subsets of the vocabulary of O_D
- EL_1,..., EL_p are *exported sameAs linkset view specifications* between E_1, ..., E_n
- ML_1, ..., ML_q are *mashup sameAs linkset view specifications* between E_1, ..., E_n
- μ is a set of *fusion rules* from O_{E1},...,O_{En} to O_D
- η is a *normalization function symbol* whose interpretation defines how to remap IRIs of the exported views to IRIs of the LDM view
- *Q* is a set of *quality assessment metrics*, which are used to quantify the quality of the Linked Data sources. This information is required by fusion rules. The specification of quality assessment metric [16] is out of the scope of this work.

The process for generating the LDM view specification λ consists of 5 steps: (1) Modeling of the mashup ontology View; (2) Generation of the exported views specifications; (3) Generation of the exported sameAs linkset view specifications; (4) Generation of the mashup sameAs linkset view specifications; (5) Definition of the normalization function, fusion rules and quality assessment metrics. Steps 2 to 5 are detailed in the following subsections.

In our framework, the materialization of a *LDM view specification* λ is automatically processed based on its specification and consists of four steps:

1. **Materialization of the exported views**. This step translates source data to the exported view vocabulary as specified by the mapping rules in the exported view specification.
2. **Materialization of the exported sameAs linksets views**. Given an exported linkset view *EL* over a Linked Data source *S*, this step exports sameAs links from *S* to the LDM view materialization.
3. **Materialization of the mashup sameAs linksets.** Given a mashup linkset view *ML*, this step computes sameAs links based on the specification of *ML*.
4. **Materialization of the mashup view**. This step materializes the mashup view by applying the normalization function and the fusion rules to the materialized exported views and the materialized sameAs Linkset views. It includes the combination and fusion of multiple representations of the same real-world object into a single representation and the resolution of data inconsistencies.

2.2 Running Example

Throughout the paper, we will adopt a simple example of a mashup application about music, called *DBB_Music*, which integrates data from two Linked Data sources: *DBpedia* [3] and *MusicBrainz* [20]. Figure 2 depicts, in UML notation, the application ontology *Music_OWL* for the *DBB_Music* mashup, which reuses terms from three well-known vocabularies: *FOAF* (Friend of a Friend), *MO* (Music Ontology) and *DC* (Dublin Core). We use the prefix *"moa:"* for the new terms defined in the *Music_OWL* ontology. *DBpedia* uses the *DBpedia Ontology [21]* which we call *DBpedia_OWL*, and we use the prefix "*dbpedia-owl*:" to refer to it. *MusicBrainz* uses the *Music Ontology* [17] and we use the prefix *"mo:"* to refer to it.

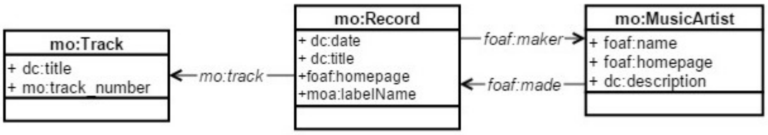

Fig. 2. Music_OWL

2.3 Specification and Materialization of Exported Views

An *exported view specification* is a quintuple *(E, S, O_S, O_E, M_E)*, where:

- *E* is the *name* of the view
- *S* is a Linked Data source
- O_S is the ontology of *S*
- O_E is the *exported view ontology* such that the terms of its vocabulary occur in the heads of the mapping rules in *M*
- M_E is a set of mapping rules from O_S to O_E

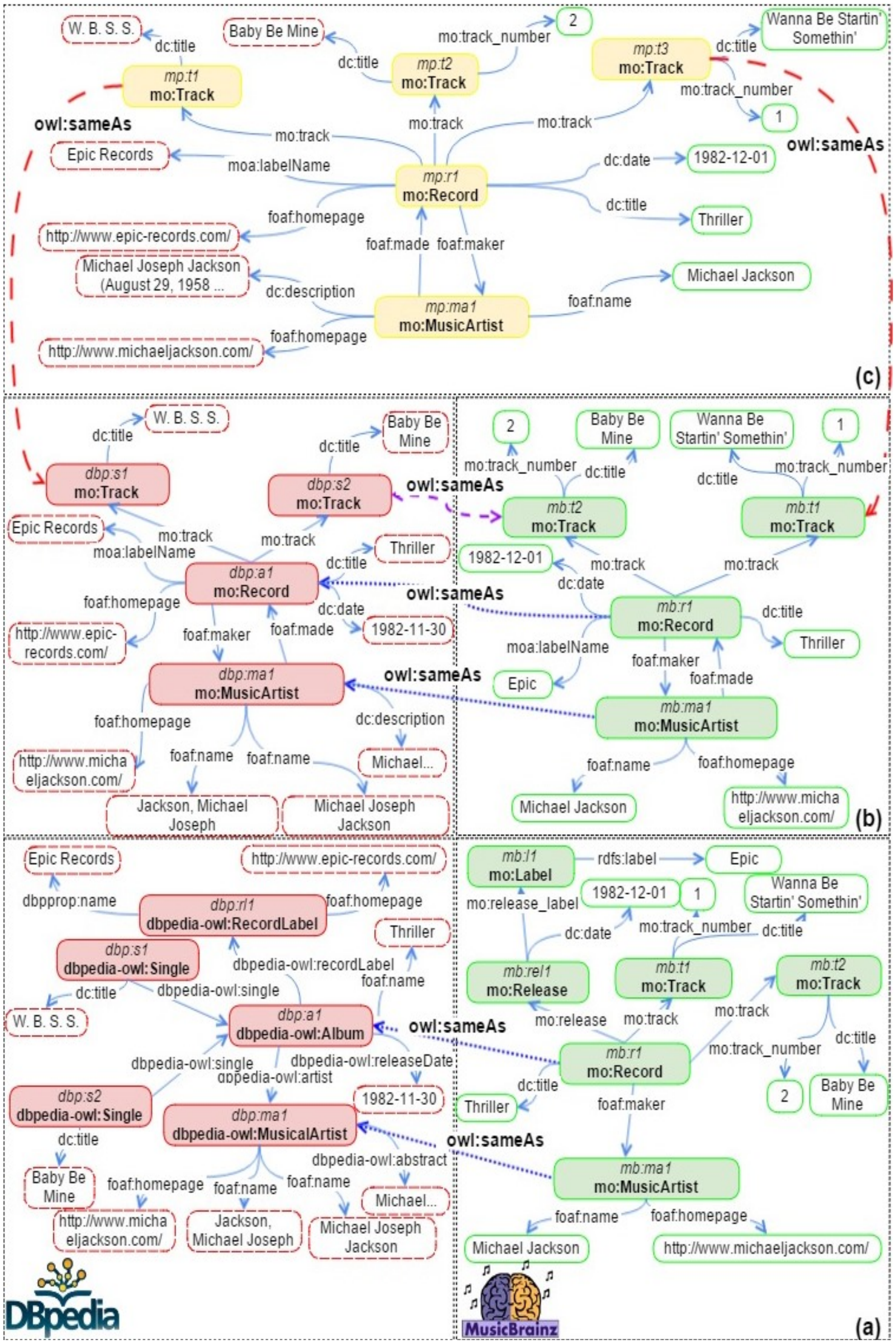

Fig. 3. (a) Data sources state; (b) Materialized Exported views and linkset views; (c) LDM view

In our framework, each linked data source exports a view whose specification is automatically generated considering the source ontology, the application ontology and the mappings between the source ontology and the application ontology. Note that, when an exported view E is part of a LDM view specification, the vocabulary of the ontology O_E must be a subset of the vocabulary of the mashup view ontology, as required in Section 2.1. The problem of generating exported views specifications is addressed in our previous work [18] and it is out of the scope of this work.

Consider the mashup application *DBB_Music* introduced in Section 2.1. An exported view specification, *DBpedia_EV*, which uses the data source *DBpedia* can be defined based on the mappings rules between *Music_OWL* and *DBpedia_OWL*. Due to space limitation, the mappings rules are omitted here. The vocabulary of the exported view ontology of *DBpedia_EV* then is:{*mo:Record*, *mo:MusicArtist*, *mo:Track*, *foaf:homepage*, *moa:labelName*, *dc:title, dc:date*, *foaf:name*, *dc:description*, *mo:track, foaf:maker, foaf:made*}

An exported view specification, *MusicBrainz_EV*, which uses the data source *MusicBrainz* can be likewise defined. The vocabulary of the exported view ontology of *MusicBrainz_EV* is:{*mo:Record*, *mo:MusicArtist*, *mo:Track*, *moa:labelName*, *dc:title, dc:date*, *foaf:name*, *foaf:homepage*, *mo:track_number*, *mo:track, foaf:maker, foaf:made*}

A materialization of an exported view requires translating source data into the exported view vocabulary as specified by the mapping rules. Referring to our case study, Figure 3(b) shows materializations of the exported views *MusicBrainz_EV* and *DBpedia_EV* obtained by applying the mappings rules.

2.4 Specification and Materialization of Exported sameAs Linkset Views

In our framework, the existing sameAs links between resources of different exported views should also be exported to the mashup view. To create exported links, the user should first specify an exported sameAs linkset view and then materialize the links.

Let $(T, S_T, O_{ST}, O_T, \mu_T)$ and $(U, S_U, O_{SU}, O_U, \mu_U)$ be two exported views. An *exported sameAs linkset view specification* is a tuple $(EL, T, U, C, C_{ST}, C_{SU})$, where:

- EL is the *name* of the view;
- $(T, S_T, O_{ST}, O_T, M_T)$ and $(U, S_U, O_{SU}, O_U, M_U)$ are exported view specifications;
- C is a class in both the vocabularies of the exported view ontologies O_T and O_U;
- C_{ST} is a class in the vocabulary of the data source ontology O_{ST} of S_T such that there is a rule in M_T, indicating that instances of C_{ST} are mapped to instances of C;
- C_{SU} is a class in the vocabulary of the data source ontology O_{SU} of S_U such that there is a rule in M_U, indicating that instances of C_{SU} are mapped to instances of C;

Let s_T and s_U be states respectively of S_T and S_U. The materialization of EL in s_T and s_U is the set $EL[s_T,s_U]$ defined as:

$$(t, owl{:}sameAs, u) \in EL[s_T, s_U] \text{ iff}$$
$$(t, owl{:}sameAs, u) \in s_T \wedge (t, rdf{:}type, C_{ST}) \in I[s_T](C_{ST}) \wedge$$
$$(u, rdf{:}type, C_{SU}) \in I[s_U](C_{SU}).$$

That is, $EL[s_T,s_U]$ imports sameAs links from s_T whose subject is in the interpretation in s_T of the class C_{ST} and whose object is in the interpretation in s_U of the class C_{SU}.

Consider, for example, the exported views *MusicBrainz_EV* and *DBpedia_EV* introduced in Section 2.3. As shown in Figure 3(a), the data source *MusicBrainz* contains sameAs links matching instances of class *mo:Record* with instances of class *dbpedia_owl:Albums*. It also contains sameAs links matching instances of *mo:MusicArtist* with *dbpedia_owl:MusicalArtist*. In order to materialize the sameAs links for the instances of the exported views *MusicBrainz_EV* and *DBpedia_EV*, two exported linkset views should be specified:

- *(EL_1, MusicBrainz_EV, DBpedia_EV, mo:Record, mo:Record, dbpedia-owl:Album)*
- *(EL_2, MusicBrainz_EV, DBpedia_EV, mo:MusicArtist, mo:MusicArtist, dbpedia-owl:MusicalArtist)*

Referring to Figure 3(a) of our case study, the sameAs link from *mb:ma1* to *dbp:ma1* is materialized by EL_2, and the sameAs link from *mb:r1* to *dbp:a1* is materialized by EL_1.

2.5 Specification and Materialization of Mashup sameAs Linkset Views

In our framework, mashup sameAs links are inferred by matching property values of resources defined in the exported views. To create mashup links, the user should first specify the mashup sameAs linkset view and then materialize the links. More precisely, a *mashup sameAs linkset view specification* is a tuple $(ML, T, U, C, p_1,\ldots, p_n, \mu)$, where:

- ML is the *name* of the view
- $(T, S_T, O_{ST}, O_T, \mu_T)$ and $(U, S_U, O_{SU}, O_U, \mu_U)$ are exported view specifications
- C is a class in both the vocabularies of the exported view ontologies O_T and O_U;
- $p_1, \ldots, p_n$ are properties of class C in both the vocabularies of the exported view ontologies O_T and O_U
- μ is a $2n$-relation, called the *match predicate*

Let e_T and e_U be states respectively of T and U. The materialization of ML in e_T and e_U is the set $ML[e_T,e_U]$ defined as:

$(t, owl{:}sameAs, u) \in ML[e_T,e_U]$ iff there are triples
$(t, rdf{:}type, C), (t, P_1, v_1),\ldots,(t, P_n, v_n) \in e_T$ and
$(u, rdf{:}type, C), (u, P_1, w_1),\ldots,(u, P_n, w_n) \in e_U$
such that $(v_1,\ldots, v_n, w_1,\ldots, w_n) \in \mu$

Consider the exported views *MusicBrainz_EV* and *DBpedia_EV* introduced in Section 3.3. Then, sameAs linkset views could be specified for matching instances of the class *mo:Track*. As an example, consider the mashup sameAs view specification for *mo:Track*: *(ML_1, MusicBrainz_EV, DBPedia_EV, mo:Track, dc:title, p)*, where the match predicate p is defined as $(v_1, w_1) \in p$ iff $\sigma(v_k, w_k) \geq \alpha$, for each $k=1$, where σ

is the 3-gram distance [14] and $\alpha = 0.5$. Referring to our case study, Figure 3(b) shows the sameAs links automatically created using ML_1.

2.6 Specification of the Normalization Function and Fusion Assertions

In this section, we first introduce the concepts of normalization function and fusion assertions, then we describe how the data fusion rules are induced from the fusion assertion.

Normalization Function. The LDM view specification includes a *normalization function*, denoted η, that remaps all IRIs in the exported views that are declared to denote the same object, via sameAs links, to one canonical target IRI. The normalization function must satisfy the following axiom (using an infix notation for sameAs):

(N$_1$) $\forall x_1 \forall x_2 (x_1 \; sameAs \; x_2 \Leftrightarrow \eta(x_1) = \eta(x_2))$

which says that the normalization function must remap to the same IRI two IRIs x_1 and x_2 iff they are declared to be equivalent via a sameAs statement of the form "x_1 *sameAs* x_2".

The normalization function partitions the IRIs of the exported views resources into a set of equivalence classes. In the materialization of the LDM view, all IRIs in the same equivalence class are homogenized by grouping all properties of those IRIs into one canonical target IRI. The canonical IRI also have owl:sameAs links pointing to the original IRIs, which makes it possible for applications to refer back to the original data sources on the Web.

Referring to our case study, the equivalence classes induced by the sameAs links in Figure 3(b) are: ε_1={*dbp:ma1*, *mb:ma1*}, ε_2={*dbp:a1*, *mb:r1*}, ε_3={*dbp:s2*, *mb:t2*}, ε_4={*dbp:s1*}, ε_5={ *mb:t1*}.

Data Fusion Assertions and Data Fusion Rules. In the rest of this section, let D be a LDM view, $O_D=(V_D, \Sigma_D)$ be the ontology of D, and η be the normalization function. Let C be a class in $\boldsymbol{V_D}$. We define

Props(C, $\boldsymbol{V_D}$) = {P / P is a property in $\boldsymbol{V_D}$ and C is a subclass of the domain of P}

In our approach, the user is free to define how to resolve the problem of contradictory attribute values when combining multiple representations of the same real-world object into a single representation (canonical IRI). This is specified with the help of *data fusion property assertions*.

A *data fusion property assertion (FPA)* for a property P in the context of a class C is an expression of form "Ψ: $P[C] \equiv f / Q$", where Ψ is the name of the *FPA*, C is a class in V_D, P and Q are properties in *Props*(C, $\boldsymbol{V_D}$), and f is a *data fusion function symbol,* which denotes a function whose domain is a set of sets of individuals and whose range is a set of individuals.

Data fusion assertions should be regarded as a shorthand notation for a class of data fusion rules. A set $\boldsymbol{A}$ of data fusion assertions defined over the vocabulary $\boldsymbol{V_D}$ of D, induces a set of fusion rules for D as follows:

- Let C be a class in $\boldsymbol{V_D}$ and P be a property in *Props*(C, $\boldsymbol{V_D}$). Assume that the assertion "Ψ: $P[C] \equiv f / Q$" is in $\boldsymbol{A}$ and that $S_1, \ldots, S_n$ are all the exported views of D

whose vocabulary contains class *C* and property *Q*. Then, the *fusion rule for P in the context C induced by* ***A*** is

$P(x,u) \leftarrow u = f(v/B[x,v])$ where:

$B[x,v] = ((C(x_1),Q(x_1,v))_{S1},\ x = \eta(x_1)); \ldots ; ((C(x_n),Q(x_n,v))_{Sn},\ x = \eta(x_n))$

The subscript S_i indicates the exported view against which $(C(x_i),\ Q(x_i,v))$ will be evaluated. Intuitively, $\{v/B[x,v]\}$ denotes the set of individuals *v* that satisfy $B[x,v]$. Then, $f(v/B[x,v])$ denotes the individual to which *f* maps the set $\{v/B[x,v]\}$.

- Let *C* be a class in $\boldsymbol{V_D}$ and *P* be a property in *Props*(*C*, $\boldsymbol{V_D}$). Assume that there is no assertion in ***A*** for property *P* in the context of *C* and that $S_1,\ldots,S_n$ are all the exported views of *D* whose vocabulary contains *C* and *P*. Then, the *fusion rule for P in the context C induced by* ***A*** is
  $P(x,v) \leftarrow ((C(x_1),P(x_1,v))_{S1},\ x = \eta(x_1)); \ldots ; ((C(x_n),P(x_n,v))_{Sn},\ x = \eta(x_n))$
- Let *C* be a class in $\boldsymbol{V_D}$. Assume that $S_1,\ldots,S_n$ are all the exported views of *D* whose vocabulary contains *C*. Then, the *fusion rule for class C induced by* ***A*** is
  $C(x) \leftarrow ((C(x_1))_{S1},\ x = \eta(x_1)); \ldots ; ((C(x_n))_{Sn},\ x = \eta(x_n))$

For example, let the *FPA* for the *moa:labelName* property in the context of class *mo:Record* be

Ψ:moa:labelName[*mo:Record*]≡*KeepSingleValueByReputation/moa:labelName*

Then, this FPA *induces* the following fusion rule:

$$\begin{aligned} &moa{:}labelName(x,u) \leftarrow u = KeepSingleValueByReputation(v/\\ &\qquad (mo{:}Record(y), moa{:}labelName(y,v))_{DBpedia\_EV}, x = \eta(y);\\ &\qquad (mo{:}Record(z), (moa{:}labelName(z,v))_{MusicBrainz\_EV}, x = \eta(z))). \end{aligned}$$

The fusion rule creates a triple of the form "*(x, moa:labelName, u)*" in a materialization of the LDM view by taking *u* as the most reputable name for *x* found in the materialization of the two exported views, *MusicBrainz_EV* and *DBpedia_EV*.

If no *FPA* was specified for the *moa:labelName* property in the context of class *mo:Record,* then the following default fusion rule would be induced:

$$\begin{aligned} &moa{:}labelName(x,v) \leftarrow (mo{:}Record(y), moa{:}labelName(y,v))_{DBpedia\_EV}, x = \eta(y));\\ &\qquad (mo{:}Record(z), (moa{:}labelName(z,v))_{MusicBrainz\_EV}, x = \eta(z))). \end{aligned}$$

This fusion rule creates a triple for each values of *moa:labelName* coming from an exported view.

2.7 Materialization of the LD Mashup View

In this section we describe how to materialize a LDM view by applying the normalization function and data fusion rules to the materialized exported views and the materialized sameAs Linkset views. The materialization process includes the combination and fusion of multiple representations of the same real-world object into a single representation and the resolution of data inconsistencies [4].

We begin by introducing the notation that will be used in this section:

- $\lambda = (D, O_D, E_1,..., E_n, EL_1,..., EL_p, ML_1,..., ML_q, \mu, \eta, Q)$ is a LDM view specification.
- *V_D is the vocabulary of O_D*
- $\boldsymbol{C} = \{ C / C$ *is a class in* $V_D \}$
- $\boldsymbol{P} = \{ P / P$ *is a property in* $V_D \}$
- $\boldsymbol{e}_1,..., \boldsymbol{e}_n$ are states of $E_1,..., E_n$ and $\boldsymbol{l}_1,..., \boldsymbol{l}_m$ are states of $EL_1,..., EL_p, ML_1,..., ML_q$, with $m=p+q$
- $\boldsymbol{s} = (\boldsymbol{e}_1,..., \boldsymbol{e}_n, \boldsymbol{l}_1,..., \boldsymbol{l}_m)$
- $IRIs(\boldsymbol{s}) = \{ x / x$ is the subject of a triple in a tripleset in $\boldsymbol{s} \}$
- $\varepsilon_1,...,\varepsilon_w$ are the equivalence classes induced by $\boldsymbol{l}_1,..., \boldsymbol{l}_m$
- $[\varepsilon_i]$ denotes the IRI that represents ε_i
- η^s denotes the interpretation of η for the $\varepsilon_1,...,\varepsilon_w$

Definition 1: $I[\boldsymbol{s}](T)$ denotes the set of triples that state $\boldsymbol{s}$ assigns to a class or property T, i.e., the interpretation of T in $\boldsymbol{s}$, defined as follows:

- Let C be a class in V_D. Assume that μ has a fusion rule whose head uses C and that the rule is of the form
  $C(x) \leftarrow (C(x_1)_{S1}, x=\eta(x_1)); ... ;(C(x_p)_{Sq}, x=\eta(x_n))$
  Then, the interpretation of C *induced by* μ *in* $\boldsymbol{s}$, also denoted $I[\boldsymbol{s}](C)$, is defined as:
  $(x, rdf{:}type, C) \in I[\boldsymbol{s}](C)$ iff $(x, rdf{:}type, C) \in I[\boldsymbol{e_1}](C), x=\eta^s(x_1)) \vee ... \vee$
  $(x, rdf{:}type, C) \in I[\boldsymbol{e_n}](C), x=\eta^s(x_n))$.
- Let P be a property in V_D. Assume that μ has a fusion rule whose head uses P and that the rule is of the form
  $P(x,v) \leftarrow (C(x_1),P(x_1, v))_{S1},x=\eta(x_1)); ... ;((C(x_n), P(x_n, v))_{Sn}, x=\eta(x_n))$.
  Then, the interpretation of P *induced by* μ *in* $\boldsymbol{s}$, also denoted $I[\boldsymbol{s}](P)$, is defined as:
  $(x, P, v) \in I[\boldsymbol{s}](P)$ iff $((x, rdf{:}type, C) \in I[\boldsymbol{e_1}](C), (x,P,v) \in I[\boldsymbol{e_1}](P), x=\eta^s(x_1)) \vee$
  $... \vee ((x, rdf{:}type, C) \in I[\boldsymbol{e_n}](C), (x,P,v) \in I[\boldsymbol{e_n}](P), x=\eta^s(x_n))$
- Let P be a property in V_D. Assume that μ has a fusion rule whose head uses P and that the rule is of the form
  $P(x, u) \leftarrow u = f(v/B[x,v])$ where:
  $B[x,v]=((C(x_1),Q(x_1,v))_{S1}, x=\eta(x_1)); ... ;((C(x_n),Q(x_n,v))_{Sn}, x=\eta(x_n))$
  Then, the interpretation of P *induced by* μ *in* $\boldsymbol{s}$, also denoted $I[\boldsymbol{s}](P)$, is defined as:
  $(x, P, u) \in I[\boldsymbol{s}](P)$ iff $u = I[\boldsymbol{s}](f)(\{v / I[\boldsymbol{s}](B[x,v])=true\})$
- If there is more than one fusion rule in μ whose head uses P, then $I[\boldsymbol{s}](P)$ is the union of all sets of triples as defined on the right-hand sides of the double implications above.

Definition 2: Recall that $\boldsymbol{C}$ is the set of all classes in V_D and $\boldsymbol{P}$ is the set of all properties in V_D and ε_i is an equivalence class induced by $\boldsymbol{l}_1,..., \boldsymbol{l}_m$. We define the *DataFusion* function as follows:

$$DataFusion(\varepsilon_i, \boldsymbol{s}) = \cup_{C \in \boldsymbol{C}} \{ (x, rdf{:}type, C) \in I[\boldsymbol{s}](C) / x = [\varepsilon_i] \} \cup$$
$$\cup_{P \in \boldsymbol{P}} \{ (x, P, y) \in I[\boldsymbol{s}](P) / x = [\varepsilon_i] \} \cup$$
$$\cup \{ ([\varepsilon_i], owl{:}sameAs, y) / y \in \varepsilon_i \wedge y \neq [\varepsilon_i] \}.$$

Definition 3: We define the *state* ***d*** or the *materialization* of *D induced by* ***s*** as (recall that $\varepsilon_1,\ldots,\varepsilon_w$ are the equivalence classes induced by $\boldsymbol{l}_1,\ldots,\boldsymbol{l}_m$):

$$\boldsymbol{d} = \cup_{i=1,\ldots,w} DataFusion(\varepsilon_i, \boldsymbol{s}).$$

Referring to our case study, Figure 3(c) shows the state of the mashup view computed from the states of the exported views and sameAs linkset views in Figure 3 (b). The mashup view has 5 resources which are computed by applying the *DataFusion* function to equivalence classes $\varepsilon_1,\ldots,\varepsilon_5$ defined in Section 2.6.

3 Incremental Maintenance of LDM Views

We now turn to the problem of maintaining a LDM view, when update operations are applied to the Linked Data sources. In this section, let:

- $\lambda = (D, O_D, E_1,\ldots, E_n, EL_1,\ldots, EL_p, ML_1,\ldots,ML_q,\mu,\eta,Q)$ be an LDM view specification
- $\boldsymbol{e}_1,\ldots, \boldsymbol{e}_n$ be states of $E_1,\ldots, E_n$ and $\boldsymbol{l}_1,\ldots, \boldsymbol{l}_m$ be states of $EL_1,\ldots, EL_p, ML_1,\ldots, ML_q$, with $m=p+q$
- $\boldsymbol{s} = (\boldsymbol{e}_1,\ldots, \boldsymbol{e}_n, \boldsymbol{l}_1,\ldots, \boldsymbol{l}_m)$
- ***d*** be the state of *D* induced by ***s***

The incremental view maintenance problem is schematically described by the diagram in Figure 4. The user specifies an update *u* against a base data source, which results in new states $\boldsymbol{e}'_1,\ldots,\boldsymbol{e}'_n$ of the exported views and new states $\boldsymbol{l}'_1,\ldots, \boldsymbol{l}'_m$ of the *sameAs* views. Let ***d'*** be the state of *D* induced by $(\boldsymbol{e}'_1,\ldots,\boldsymbol{e}'_n, \boldsymbol{l}'_1,\ldots,\boldsymbol{l}'_m)$. We say that a set of updates U_D over the state ***d*** correctly maintains *D* iff $U_D(\boldsymbol{d}) = \boldsymbol{d}'$.

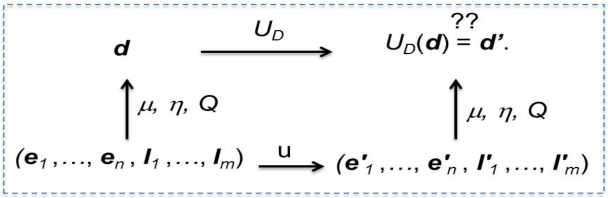
$$\begin{tikzcd}
\boldsymbol{d} \arrow[r, "U_D"] & U_D(\boldsymbol{d}) \overset{??}{=} \boldsymbol{d}'. \\
(\boldsymbol{e}_1,\ldots,\boldsymbol{e}_n,\boldsymbol{l}_1,\ldots,\boldsymbol{l}_m) \arrow[u, "{\mu, \eta, Q}"'] \arrow[r, "u"] & (\boldsymbol{e}'_1,\ldots,\boldsymbol{e}'_n,\boldsymbol{l}'_1,\ldots,\boldsymbol{l}'_m) \arrow[u, "{\mu, \eta, Q}"']
\end{tikzcd}$$

Fig. 4. Incremental View Maintenance Problem

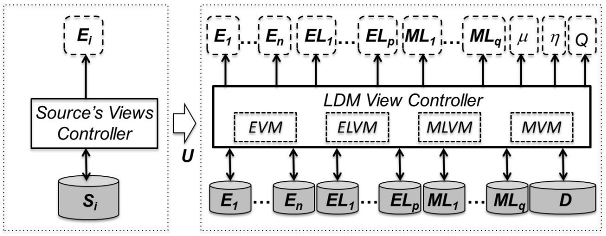
E_i
Source's Views Controller
S_i
U
E_1 … E_n EL_1 … EL_p ML_1 … ML_q μ η Q
LDM View Controller
EVM
ELVM
MLVM
MVM
E_1 … E_n EL_1 … EL_p ML_1 … ML_q D

Fig. 5. Suggested Platform for LDM View Maintenance

Figure 5 shows the main components of the architecture we suggest to incrementally maintain the LDM view *D*. For each data source $\boldsymbol{S_i}$ that exports a view $\boldsymbol{E_i}$ to *D*, there is a *Source View Controller*, with the following functionality:

1. Identify updates on $\boldsymbol{S_i}$ that are relevant to *D*, that is, relevant to exported views or materialized linkset views used to construct *D*. This is computed from the specification of the exported views and linkset views.
2. For each relevant update *u*, create the set: $R = \{r \,/\, r$ is the IRI of a resource affected by $u\}$.
3. Send *R* to the LDM View Controller.

Note that *R* can be automatically computed based on the exported and linkset views specifications. Hence, no access to the mashup view is required. The problem of computing *R* is out of the scope of this paper. A similar problem was addressed in a previous work [22].

The *LDM View Controller* receives *R* and then performs the incremental maintenance of the exported and linkset views followed by the incremental maintenance of mashup view *D*. The *LDM View Controller* has 4 main components (see Figure 5):

- ***EVM Module***: Maintains the exported views $E_1,\ldots, E_n$.
- ***ELVM Module:*** Maintains the exported linkset views $EL_1,\ldots, EL_p$.
- ***MLVM Module:*** Maintains the mashup linkset views $ML_1,\ldots, ML_q$.
- ***MVM Module:*** Maintains the mashup view *D*.

The incremental maintenance of linkset views has already been addressed in our previous work [5], and the problem of incremental maintenance of exported views is very similar to the problem addressed in [22]. Therefore, we do not address those problems in this paper.

In order to accomplish incremental maintenance of mashup view *D*, the ***MVM*** module executes the procedure *Effect*(*R*) considering $\boldsymbol{t} = (\boldsymbol{e'_1},\ldots,\boldsymbol{e'_n}, \boldsymbol{l'_1},\ldots, \boldsymbol{l'_m})$, the new states of the exported views and sameAs views. The definition of *Effect* depends on the following definitions:

Definition 4: Let *r* be a resource in *R*, $\boldsymbol{DA}(r) = \boldsymbol{A_{OLD}}(r) \cup \boldsymbol{A_{NEW}}(r)$ where:
$A_{OLD}(r) = \{x \in IRIs(\boldsymbol{t}) / (\exists y \in IRIs(\boldsymbol{d}))((y, owl{:}sameAs, r) \in \boldsymbol{d} \; @ \; (y, owl{:}sameAs, x) \in \boldsymbol{d})\}$
$A_{NEW}(r) = \{x \in IRIs(\boldsymbol{t}) / \eta[\boldsymbol{t}](x) = \eta[\boldsymbol{t}](r)$.

Definition 5: $\boldsymbol{DA}(R) = \cup_{r \in R} \boldsymbol{DA}(r)$.

Definition 6: $\boldsymbol{DA^*}(R) = \cup_{r \in R} \boldsymbol{DA^*}(r)$ where
$\boldsymbol{DA}^*(r) = \{ x \in IRIs(d) / (\exists y \in \boldsymbol{DA}(r))((x, owl{:}sameAs, y) \in d)\}$.

Definition 7: $\boldsymbol{IA^*}(R) = (\cup_{r \in R} \boldsymbol{IA^*}(r)) - \boldsymbol{DA^*}(R)$ where,
$\boldsymbol{IA}^*(r) = \{ x \in IRIs(d) / x \notin \boldsymbol{DA}^*(r) \wedge (\exists y \in \boldsymbol{DA}^*(r))((x, P, y) \in d)\}$.

The procedure *Effect* in Table 1, computes the mashup view maintenance updates in 2 steps:

<u>Step 1</u>: Computes the new state of the mashup resources in $\boldsymbol{DA}^*(R)$, i.e. directly affected by the resources in *R*.
<u>Step 2</u>: Updates the state of the mashup resources in $\boldsymbol{IA}^*(\mathrm{R})$, i.e. indirectly affected by the updates in step 1.

Table 1. Procedure *Effect*(*R*)

```
Parameters (as defined above):
  $\lambda$, the LDM view specification
  $t = (e'_1,\ldots, e'_n, l'_1,\ldots, l'_m)$
  $d$, the state of mashup view $D$ induce by $s$

Input: $R$, the set of IRIs of resources affected by an update

Step 1. Compute the new state of the mashup resources in $DA^*(R)$.
1.1 Compute $DA(R)$ and $DA^*(R)$; /* See Definitions 4-6
1.2. Retrieve $\Delta_{OLD}$ the old states of the mashup resources in $DA^*(R)$;
    $\Delta_{OLD} := \cup_{x \in DA^*(R)} I[d](x)$, where $I[d](x)$ denotes the state of the resource $x$ in state $d$,
        i.e., the set of triples in $d$ where the subject of those triple is $x$.
1.3. Compute $\Delta_{NEW}$ the new states of the mashup resources for the resources in $DA(R)$;
    $\Delta_{NEW} := \cup_{[\varepsilon] \in S} DataFusion(\varepsilon, t)$ where $S = \{ \eta[t](x) / x \in DA(R)\}$,
        i.e., the set of equivalence classes for the resources in $DA(R)$.
1.4. $d := (d - \Delta_{OLD}) \cup \Delta_{NEW}$;

Step 2. Update the state of the mashup resources in $IA^*(R)$.
 2.1 Compute $IA^*(R)$; /* see Definition 7.
 2.2 For each $m$ in $IA^*(R)$ do {
    2.2.1 $AP(m) = \{ P / (\exists y \in DA^*(R)) \wedge (m, P, y) \in d \}$;
     /*$AP(m)$ denotes the set of properties relating $m$ with mashup resources in $DA^*(R)$.
    2.2.2. Computes the new states for the properties in $AP(m)$;
      For each $P$ in $AP(m)$ do {
        $\Delta_{OLD} := \{(m, P, y) / (\exists y \in d) \wedge (m, P, y) \in d\}$;
        $\Delta_{NEW} := I[t](P)$; /* see Definition 1.
        $d := (d - \Delta_{OLD}) \cup \Delta_{NEW}$;
```

To illustrate this strategy, let *u* be the following update on *DBpedia:*

```
WITH <http://dbpedia.org>
DELETE { ?x dc:title "W. B. S. S."  }
INSERT { ?x dc:title "Wanna Be Startin' Somethin'" }
WHERE{?x rdfs:type dbpedia-owl:Single.?x dc:title "W.B.S.S"}
```

Phase 1: (Executed by the *DBpedia* View Controller)
Considering the state of DBpedia in Figure 3(a), the update *u* changes the title of the instance ***dbp:s1*** to "Wanna Be Startin' Somethin'". Therefore, the *DBpedia* View Controller sends *R*={*dbp:s1*} to the *LDM View Controller.*

Phase 2: (Executed by the LDM View Controller)
The *LDM View Controller* receives *R* and then performs the incremental maintenance of the exported views and linkset views. Based on ML_1 specification, the resources *dbp:s1* and *mb:t1* are computed as equivalent. Therefore a new sameAs link (*dbp:s1, owl:sameAs, mb:t1*) is added to ML_1.

Then, the procedure *Effect*({*dbp:s1*}) computes the mashup view maintenance updates in 2 steps:

Step 1: Compute the new state of the mashup resources for the resources in ***DA****({*dbp:s1*})
Step 1.1: Considering the state ***t*** and ***d*** in Figures 3(b) and 3(c) we have that:
DA({*dbp:s1*})={*dbp:s1, mb:t1*}
DA*({*dbp:s1*})={*mp:t1, mp:t3*}
Step 1.2: $\Delta_{OLD} = I[\boldsymbol{d}](mp{:}t1) \cup I[\boldsymbol{d}](mp{:}t3)$
Step 1.3: Δ_{NEW} = *DataFusion*(ε, ***t***) where ***ε*** ={*dbp:s1, mb:t1*}
Step 1.4: $\boldsymbol{d} = (\boldsymbol{d} - \Delta_{OLD}) \cup \Delta_{NEW}$
Step 2. Update the state of the mashup resources in $\boldsymbol{IA}^*$({*dbp:s1*})
Step 2.1. $\boldsymbol{IA}^*$({*dbp:s1*})={*mp:r1*}
Step 2.2. Computes the new states for the properties in ***AP***(*mp:r1*)
Step 2.2.1 ***AP***(*mp:r1*) = {*mo:track*}
Step 2.2.2 Δ_{OLD}={(*mp:r1, mo:track, mp:t1*), (*mp:r1, mo:track, mp:t3*)}
Δ_{NEW} = {(*mp:r1, mo:track, mp:t4*)}
$\boldsymbol{d} := \boldsymbol{d} - \Delta_{OLD} \cup \Delta_{NEW}$

4 Related Work

The problem of incremental view maintenance has been extensively studied in the literature. However, for the most part, views are defined over a single data source, e.g., for relational views [6], for object-oriented views [15], for semi-structured views [1], for XLM Views [7], [23], and for RDF views [22]. None of the proposed techniques can be directly applied to data integration views.

The incremental maintenance of data integration views is address in [9], [10], which focus on the relational views. In [10], an algorithm for the incremental maintenance of outerjoin and match views was developed. In [9], algebraic change propagation algorithms were developed for the maintenance of the outerjoin view. Despite of their important contributions, none of those techniques can be applied to LDM views, since the nature of linked data sources poses new challenges for dealing with the data fusion problem, specially the treatment of sameAs.

Recently, two frameworks were proposed for creating LDM views. The ODCleanStore framework [12] offers Linked Data fusion, dealing with inconsistencies. LDIF - Linked Data Integration Framework [19] implements a mapping language, deals with URIs remapping and uses named graphs for registering data provenance.

To the best of our knowledge, incremental maintenance of LDM views has not yet been addressed in any framework.

5 Conclusions

In this paper, we first proposed an ontology-based framework for specifying Linked Data Mashup views. In the framework, a Linked Data mashup view is formally specified with the help of exported views, linkset views, data fusion function and normalization function. The LDM view specification is used to automatically materialize the mashup view. Then, we outlined a strategy that uses the mashup view specification for incrementally maintain the mashup view.

Our strategy addressed the problem of dealing with changes in the set of sameAs links between IRIs from different data sources and with the very question of recomputing mashup property values, in the presence of updates on the data sources.

Acknowledgments. This work was partly funded by CNPq, under grants 248987/2013-5, 442338/2014-7 and 303332/2013-1, by FAPERJ, under grant E-26/201.337/2014, and by CAPES, under grants PROCAD/NF 789/2010, 1410827.

References

1. Abiteboul, S., McHugh, J., Rys, M., Vassalos, V., Wiener, J.L.: Incremental Maintenance for Materialized Views over Semistructured Data. VLDB 1998, 38–49 (1998)
2. Berners-Lee, T.: Linked Data (2006). http://www.w3.org/DesignIssues/LinkedData.html
3. Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., Hellmann, S.: DBpedia - A crystallization point for the Web of Data. J. Web Semant. 7(3), 154–165 (2009)
4. Bleiholder, J., Naumann, F.: Data fusion. ACM Comput. Surv. 41(1), 1:1–1:41 (2009)
5. Casanova, M.A., Vidal, V.M., Lopes, G.R., Leme, L.A.P., Ruback, L.: On materialized sameAs linksets. In: Decker, H., Lhotská, L., Link, S., Spies, M., Wagner, R.R. (eds.) DEXA 2014, Part I. LNCS, vol. 8644, pp. 377–384. Springer, Heidelberg (2014)
6. Ceri, S., Widom, J.: Deriving productions rules for incremental view maintenance. VLDB 1991, 577–589 (1991)
7. Dimitrova, K., El-Sayed, M., Rundensteiner, E.A.: Order-sensitive View Maintenance of Materialized XQuery Views. ER 2003, 144–157 (2003)
8. Endres, B.N.: Semantic Mashups. Springer, Heidelberg (2013)
9. Griffin, T., Libkin, L.: Algebraic change propagation for semijoin and outerjoin queries. SIGMOD Record 27(3) (1998)
10. Gupta, A., Mumick, I.S.: Materialized Views. MIT Press (2000)
11. Hanh, H.H., Tai, N.C., Duy, K.T., Dosam, H., Jason, J.J.: Semantic Information Integration with Linked Data Mashups Approaches. Int. J. Distrib. Sens. N. (2014)
12. Knap, T., Michelfeit, J., Daniel, J., Jerman, P., Rychnovsky, D., Soukup, T., Necasky, M.: ODCleanStore: A Framework for Managing and Providing Integrated Linked Data on the Web. In: Sean Wang, X., Cruz, I., Delis, A., Hua, G. (eds.) Web Information Systems Engineering - WISE 2012. LNCS, vol. 7651, pp. 815–816. Springer, Heidelberg (2012)
13. Kobilarov, G., Scott, T., Raimond, Y., Oliver, S., Sizemore, C., Smethurst, M., Bizer, C., Lee, R.: Media Meets Semantic Web – How the BBC Uses DBpedia and Linked Data to Make Connections. In: Aroyo, L., Traverso, P., Ciravegna, F., Cimiano, P., Heath, T., Hyvönen, E., Mizoguchi, R., Oren, E., Sabou, M., Simperl, E. (eds.) ESWC 2009. LNCS, vol. 5554, pp. 723–737. Springer, Heidelberg (2009)
14. Kondrak, G.: N-Gram similarity and distance. In: Consens, M.P., Navarro, G. (eds.) SPIRE 2005. LNCS, vol. 3772, pp. 115–126. Springer, Heidelberg (2005)

15. Kuno, H.A., Rundensteiner, E.A.: Incremental Maintenance of Materialized Object-Oriented Views in MultiView: Strategies and Performance Evaluation. IEEE TDKE 10(5), 768–792 (1998)
16. Mendes, M., Mühleisen, H., Bizer, C.: Sieve: Linked Data Quality Assessment and Fusion. Invited paper at the LWDM 2012 (2012)
17. Raimond, Y., Abdallah, S., Sandler, M., Giasson, F.: The Music Ontology. In: International Conference on Music Information Retrieval, pp. 417–422 (2007)
18. Sacramento, E.R., Vidal, V.M.P., Macedo, J.A.F., Lóscio, B.F., Lopes, F.L.R., Casanova, M.A.: Towards Automatic Generation of Application Ontologies. JIDM 1(3), 535–550 (2010)
19. Schultz, A., Matteini, A., Isele, R., Mendes, P., Bizer, C., Becker, C.: LDIF - A Framework for Large-Scale Linked Data Integration. In: WWW2012, Developers Track (2012)
20. Swartz, A.: MusicBrainz: A Semantic Web Service. IEEE Intelligent Systems 17(1), 76–77 (2002)
21. The DBpedia Ontology (2014). http://wiki.dbpedia.org/Ontology2014
22. Vidal, V.M.P., Casanova, M.A., Cardoso, D.S.: Incremental Maintenance of RDF Views of Relational Data. In: Meersman, R., Panetto, H., Dillon, T., Eder, J., Bellahsene, Z., Ritter, N., De Leenheer, P., Dou, D. (eds.) ODBASE 2013. LNCS, vol. 8185, pp. 572–587. Springer, Heidelberg (2013)
23. Vidal, V.M.P., Lemos, F.C.L., Araújo, V., Casanova, M.A.: A Mapping-Driven Approach for SQL/XML View Maintenance. ICEIS 2008, 65–73 (2008)