

# On Materialized sameAs Linksets

Marco A. Casanova<sup>1</sup>, Vânia M. P. Vidal<sup>2</sup>, Giseli Rabello Lopes<sup>1</sup>,  
Luiz André P. Paes Leme<sup>3</sup>, Livia Ruback<sup>1</sup>

<sup>1</sup>Department of Informatics, PUC-Rio, Rio de Janeiro, RJ – Brazil  
{casanova, grlopes, lrodrigues}@inf.puc-rio.br

<sup>2</sup>Computer Science Department, UFC, Fortaleza, CE – Brazil  
vvidal@lia.ufc.br

<sup>3</sup>Computer Science Institute, UFF, Niterói, RJ – Brazil  
lapaesleme@ic.uff.br

**Abstract.** The Linked Data initiative promotes the publication of previously isolated databases as interlinked RDF datasets, thereby creating a global scale data space. Datasets are frequently interlinked using some automated matching process that results in a *materialized sameAs linkset*, that is, a set of links of the form  $(s, owl:sameAs, o)$ , which asserts that  $s$  denotes the same resource as  $o$ . This paper proposes strategies to reduce the cognitive overhead of creating materialized sameAs linksets and to correctly maintain them. The paper also outlines an architecture to improve the support for materialized sameAs linksets.

**Keywords:** RDF dataset interlinking, Linked Data, view update.

## 1 Introduction

The Linked Data initiative [2] promotes the publication of previously isolated databases as interlinked RDF datasets, or simply *datasets*, thereby creating a global scale data space, known as the Web of Data. Therefore, at the heart of the Linked Data initiative stands the problem of interlinking datasets.

Datasets are frequently interlinked using some automated matching process, whose result is a *materialized linkset*, that is, a set of links that is automatically created and explicitly stored. Of special interest is the case of *materialized sameAs linksets* consisting of links of the form  $(s, owl:sameAs, o)$ , which asserts that  $s$  denotes the same resource as  $o$ . In this paper, we propose strategies to reduce the cognitive overhead of creating materialized sameAs linksets and to correctly maintain them. The paper also outlines how triple stores could improve the support for materialized sameAs linksets.

Consider a materialized sameAs linkset  $L$  interlinking datasets  $T$  and  $U$ . To simplify the creation of  $L$ , we advocate that the administrator of each dataset, who understands how the dataset is organized, should pre-define views that act as resource catalogues – sets of resources with useful properties. The task of the user starts by selecting a view  $\nu$  over  $T$  and a view  $w$  over  $U$ , that define catalogues of comparable resources, and using  $\nu$  and  $w$  to create  $L$ . We therefore suggest replacing the difficulties of understanding how  $T$  and  $U$  are modeled by selecting pre-defined views.

To address the maintenance of  $L$ , we propose an incremental strategy, justified for two reasons: (1)  $L$  is computed by a (complex) instance matching process between property values obtained from  $T$  and  $U$  using views; and (2)  $L$  does not contain the property values that generated the sameAs links. The architecture to support materialized sameAs linksets we suggest separates the problem of detecting updates on the datasets in a *view controller* component from the problem of maintaining the materialized sameAs linkset, which is the responsibility of a *linkset controller*.

Briefly, as for related work, several tools have been specifically designed to create linksets [3][8][13]. Tools have also been developed to recommend datasets with a high probability of interlinking [4][5][7], thereby reducing the cost of the interlinking process. The introduction of views, as suggested in Section 3, would simplify the configuration of the tools designed to create links. Furthermore, the link recommendations tools would benefit from the publication of view metadata. In another direction, tools, such as DSNotify [10], have been designed to inform database administrators about dataset changes and to allow them to preserve link integrity. DSNotify is closely related to the *view controller* component discussed in Section 4. Finally, it has been shown that incremental maintenance generally outperforms full view re-computation [11][12]. In fact, we argued that the linkset maintenance problem might be addressed in much the same way as the materialized view maintenance problem.

The paper is organized as follows. Section 2 introduces basic concepts and a simple example used in the paper. Section 3 covers how to create materialized sameAs linksets. Section 4 discusses how to maintain sameAs linksets and suggests how triple stores could better support them. Finally, Section 5 contains the conclusions.

## 2 Basic Concepts

An *RDF dataset*, or simply a *dataset*, is a set of RDF triples. A resource identified by an IRI (*Internationalized Resource Identifier*)  $s$  is *defined in* a dataset  $T$  iff  $s$  occurs as the subject of a triple in  $T$ . A triple in  $T$  *defines a property of*  $s$  iff the triple is of the form  $(s, p, v)$  and  $p$  is not *rdf:type*.

Let  $T$  and  $U$  be two datasets. A *link* from  $T$  to  $U$  is an RDF triple  $(s, p, o)$  such that  $s$  is defined in  $T$ ,  $o$  is defined in  $U$  and  $p$  is not *rdf:type*. We say that  $T$  is *linked to*  $U$ , or that  $U$  is *linked from*  $T$ , iff there is at least a link from  $T$  to  $U$ . A *linkset* from  $T$  to  $U$  is a set of links from  $T$  to  $U$ . A *sameAs link* is a link of the form  $(s, owl:sameAs, o)$ , which asserts that  $s$  denotes the same resource as  $o$ .

We will adopt a simple example based on two datasets. The first one, the *Lattes dataset*, represents CVs of Brazilian researchers and was extracted from the Lattes platform. We refer to this dataset as *BrCV* and assume that it is available at `http://lattes.br/` (a fictitious IRI). The second dataset, the *Semantic Web Conference Corpus*, contains triples about the main conferences and workshops in the area of Semantic Web research. We refer to this dataset as *SWCC* and assume that it is available at `http://data.semanticweb.org/`. *BrCV* uses a specific ontology, which we call the *Lattes ontology*, and *SWCC* uses the *Semantic Web Conference (SWC) ontology* [6].

A SPARQL query  $F$  is a *simple property path query*, or a *simple query*, iff

- The CONSTRUCT clause of  $F$  has exactly one template of the form “ $?x \text{ rdf:type } C$ ” and a list of templates of the form “ $?x P_k ?p_k$ ”, where  $C$  is a class and  $P_k$  is a property, for  $k=1, \dots, n$ ; we say that  $V_F = \{C, P_1, \dots, P_n\}$  is the *vocabulary* of  $F$ .
- $F$  contains a single FROM clause, specifying the dataset used to evaluate  $F$ .
- The WHERE clause of  $F$  is a list “ $c, p_1, \dots, p_m, f_1, \dots, f_n$ ”, where
  - $c$  is of the form “ $?x \text{ rdf:type } D$ ”, where  $C$  and  $D$  are not necessarily equal
  - for each  $k=1, \dots, m$ , the expression  $p_k$  is a sequence property path of the form “ $?x R_k^1 / \dots / R_k^{n_k} ?p_k$ ”, where  $R_k^i$  is an IRI or the inverse of a path consisting of a single IRI, for  $i=1, \dots, n_k$
  - for each  $l=1, \dots, n$ , the expression  $f_l$  is a SPARQL filter restricting the variables used in the path expressions.

Let  $T$  be the dataset specified in the FROM clause of  $F$ . When evaluated against  $T$ , the simple query  $F$  returns a set of triples, which we denote  $F[T]$ .

A *simple view definition* is a pair  $\mathbf{v} = (V_F, F)$ , where

- $F$  is a simple SPARQL query, called the *view mapping*, whose FROM clause specifies the dataset over which  $\mathbf{v}$  is evaluated.
- $V_F$  is the vocabulary of  $F$ , also called the *view vocabulary* (hence,  $V_F$  consists of a single class and an ordered list of properties).

A *materialization* of  $\mathbf{v}$  is the process of computing the set  $F[T]$  and explicitly storing it as part of a dataset.

A *linkset view definition* is a quintuple  $\mathbf{l} = (p, F, G, \pi, \mu)$ , where

- $p$  is an object property
- $F$  and  $G$  are simple queries whose vocabularies have the same cardinality  $n$  and whose FROM clauses specify the datasets over which  $\mathbf{l}$  is evaluated
- $\pi$  is a permutation of  $(1, \dots, n)$ , called the *alignment* of  $\mathbf{l}$
- $\mu$  is a  $2n$ -relation, called the *match predicate* of  $\mathbf{l}$

Let  $V_F = \{C, P_1, \dots, P_n\}$  be the vocabulary of  $F$  and  $V_G = \{D, Q_1, \dots, Q_n\}$  be the vocabulary of  $G$ . Intuitively,  $\pi$  indicates that, for each  $k=1, \dots, n$ , the match predicate will compare values of  $P_k$  with values of  $Q_m$ , where  $m = \pi(k)$ .

We also admit a linkset view definition of the form  $\mathbf{l} = (p, \mathbf{v}, \mathbf{w}, \pi, \mu)$ , where  $\mathbf{v} = (V_F, F)$  and  $\mathbf{w} = (V_G, G)$  are simple view definitions. This alternative form translates to the previous one, if we replace  $\mathbf{v}$  and  $\mathbf{w}$  by their queries,  $F$  and  $G$ , respectively.

Let  $T$  and  $U$  be the datasets specified in the FROM clauses of  $F$  and  $G$ , respectively. We say that  $\mathbf{l}$  is *evaluated over  $T$  and  $U$*  and that  $\mathbf{l}$  is *from  $T$  to  $U$* . The linkset view definition  $\mathbf{l}$  induces a set of triples, denoted  $\mathbf{l}[T, U]$ , as follows:

$(s, p, o) \in \mathbf{l}[T, U]$  iff there are triples  $(s, \text{rdf:type}, C), (s, P_1, s_1), \dots, (s, P_n, s_n) \in F[T]$  and  $(o, \text{rdf:type}, D), (o, Q_1, o_1), \dots, (o, Q_n, o_n) \in G[U]$  such that  $(s_1, \dots, s_n, o_{m_1}, \dots, o_{m_n}) \in \mu$  where  $m_k = \pi(k)$ , for each  $k=1, \dots, n$

A *materialization* of  $\mathbf{l}$  is the process of computing the set  $\mathbf{l}[T, U]$  and explicitly storing it as part of a dataset. Again, we could expand the abstract notation to indicate the dataset and provide a name for the materialization of a linkset view definition.

### 3 Creating Materialized sameAs Linksets

To create a materialized sameAs linkset  $L$  between  $T$  and  $U$ , the user should:

- (SA1) Specify a sameAs linkset view definition over  $T$  and  $U$ .
- (SA2) Materialize  $L$ .

Step (SA1) requires that the user understand how datasets  $T$  and  $U$  are modeled, which properties may act as identifiers for the classes, etc. Based on this analysis, the user will create queries  $F$  and  $G$  (an example is given below). Step (SA2) is usually costly and may return only approximate values.

Using our running example, assume that the user wants to create a sameAs linkset, *sameRe*, between researchers in the *SWCC* dataset and those represented in the *BrCV* dataset by their CVs. Then, he may match the person's name from both datasets and, to disambiguate, use the homepage of the organization the person works for. The user starts by specifying a sameAs linkset view definition  $l = (owl:sameAs, F_{SWCC}, G_{BrCV}, \pi, \mu)$ , where  $F_{SWCC}$  is the SPARQL query

```

1. PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2. PREFIX foaf: <http://xmlns.com/foaf/0.1/>
3. CONSTRUCT { ?x rdf:type foaf:Person .
4.             ?x foaf:firstName ?fn .
5.             ?x foaf:lastName ?ln .
6.             ?x foaf:workplaceHomepage ?op }
7. FROM <http://data.semanticweb.org/>
8. WHERE
9. { ?x rdf:type foaf:Person .
10.  ?x foaf:firstName ?fn .
11.  ?x foaf:lastName ?ln .
12.  ?x foaf:member/foaf:page ?op }

```

$G_{BrCV}$  is the SPARQL query

```

13. PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
14. PREFIX foaf: <http://xmlns.com/foaf/0.1/>
15. PREFIX la: <http://onto.lattes.br/>
16. CONSTRUCT { ?x rdf:type foaf:Person .
17.             ?x foaf:firstName ?fn .
18.             ?x foaf:lastName ?ln .
19.             ?x foaf:workplaceHomepage ?op }
20. FROM <http://lattes.br/>
21. WHERE
22. { ?x rdf:type la:Curriculum .
23.  ?x foaf:firstName ?fn .
24.  ?x foaf:lastName ?ln .
25.  ?x la:refersToWorkedFor/la:refersToOrg/foaf:homepage ?op }

```

the alignment  $\pi$  is the identity permutation and the match predicate  $\mu$  is defined as

$$(s_1, \dots, s_n, o_1, \dots, o_n) \in \mu \text{ iff } \sigma(s_k, o_k) \geq \alpha, \text{ for each } k=1, \dots, n$$

where  $\sigma$  is the 3-gram distance and  $\alpha = 0.5$ .

The strategy we propose to reduce the cognitive overhead of creating materialized sameAs linksets rests on the idea that the responsibility for Step (SA1) should not be placed on the user that is trying to create the linkset, but it should be shared with the administrator of each dataset. That is, the administrator should publish one or more view definitions over the dataset such that

- (V1) Each view definition is simple.
- (V2) Each view definition includes a mapping to the underlying dataset.
- (V3) Each view is accompanied by metadata that describe the set of instances represented in the view (using the VoID vocabulary 1, for example) and indicate how its vocabulary is pre-aligned with standard vocabularies.

Thus, a user who wishes to create links between  $T$  and  $U$  may browse the published metadata to find simple view definitions  $\mathbf{v}=(V_F,F)$  and  $\mathbf{w}=(V_G,G)$ , thereby simplifying Step (SA1). As an example, consider again the problem of creating a sameAs linkset, *sameRe*, between researchers represented in the *SWCC* dataset and researchers represented in the *BrCV* dataset by their CVs. For disambiguation purposes, one identifies a researcher by his name and the homepage of the organization he works for. The creation of the *sameRe* linkset will be considerably simplified if the administrators of the *SWCC* and *BrCV* datasets define views that capture the properties that qualify researchers in their datasets.

For example, the administrator of *SWCC* may create a simple view definition  $ReSWCC = (V_F, F_{SWCC})$ , where  $V_F = \{foaf:Person, foaf:firstName, foaf:lastName, foaf:workplaceHomepage\}$  and  $F_{SWCC}$  is the query already used at the beginning of this section. Likewise, the administrator of *BrCV* may create a simple view definition  $ReBrCV = (V_G, G_{BrCV})$ , where  $V_G = \{foaf:Person, foaf:firstName, foaf:lastName, foaf:workplaceHomepage\}$  and  $G_{BrCV}$  is the SPARQL query previously defined. Lastly, the user creates the sameAs linkset view definition  $I = (owl:sameAs, ReSWCC, ReBrCV, \pi, \mu)$ , as before, except that he will use the view definitions and will trivially specify the alignment  $\pi$  as the identity permutation. This simple example illustrates how view definitions help define materialized linksets.

## 4 Maintaining Materialized sameAs Linksets

We may address the maintenance of materialized sameAs linksets using strategies similar to those devised to maintain materialized views: *rematerialize*  $L$ , when updates are applied to  $T$  or  $U$ ; *incrementally maintain*  $L$ , based on the updates on  $T$  or  $U$ ; *invalidate* links in  $L$  that are affected by updates on  $T$  or  $U$ . We do not consider versioning  $L$  since we would have to assume that  $T$  and  $U$  are also versioned, which leads a different set of problems. To decide which is best among these alternatives, observe that: (1)  $L$  is computed by a (potentially complex) matching process between property values obtained from  $T$  and  $U$  using queries; and (2)  $L$  does not contain the triples capturing the property values that generated the sameAs links.

Hence, by (1), rematerialization is potentially costlier than incremental maintenance or link invalidation. But, in either case, by (2), it is impossible to maintain or invalidate links in  $L$  in the presence of updates on  $T$  and  $U$ . Indeed, by (2),  $L$  does not contain enough information that would permit: (i) detecting when an update on  $T$  or  $U$  invalidates a sameAs link in  $L$ ; (ii) recomputing a sameAs link after an update on  $T$  or  $U$  occurs.

Suppose that  $L$  is obtained by the materialization of  $I = (owl:sameAs, \mathbf{v}, \mathbf{w}, \pi, \mu)$ , where  $\mathbf{v}=(V_F,F)$  and  $\mathbf{w}=(V_G,G)$  are view definitions over  $T$  and  $U$ , respectively. To

incrementally maintain  $L$ , we have to analyze how updates on  $T$  and  $U$  affect  $L$  and to compute the changes that must be applied to  $L$ .

Let  $u$  be an update on  $T$  (the discussion for updates on  $U$  is entirely similar). Let  $T_{old}$  and  $T_{new}$  be the states of  $T$  before and after the update. The strategy we adopt to incrementally maintain  $L$  briefly goes as follows:

- (1) (Before  $u$  is executed) Compute the set  $\mathbf{R}$  of resources defined in  $F[T_{old}]$  whose properties in  $F[T_{old}]$  are affected by  $u$ .
- (2) (After  $u$  is executed) For each  $s \in \mathbf{R}$ ,
  - a. Delete from  $L$  all sameAs links of the form  $(s, owl:sameAs, o)$ , for some  $o$ .
  - b. If  $s$  is still defined in  $F[T_{new}]$ , then
    - i. Retrieve the (new) properties of  $s$  defined in  $F[T_{new}]$ .
    - ii. Try to match the (new) properties of  $s$  with those of a resource  $o$  in  $G[U]$ ; if a match is found, add  $(s, owl:sameAs, o)$  to  $L$ .

We illustrate this strategy with the example of Section 3, which creates a sameAs linkset, *sameRe*, as a materialization of  $I = (owl:sameAs, ReSWCC, ReBrCV, \pi, \mu)$ .

**Step 1.** As an example, suppose that  $u$  is the following update on *SWCC*:

```
1. WITH <http://data.semanticweb.org>
2. DELETE { ?x foaf:page "www.pucrio.br" }
3. INSERT { ?x foaf:page "www.puc-rio.br" }
4. WHERE
5. { ?x foaf:page "www.pucrio.br" }
```

The set  $\mathbf{R}$  is the result of the following query  $F_R$ , synthesized from  $F_{SWCC}$  and  $u$ :

```
6. SELECT ?x
7. FROM <http://data.semanticweb.org>
8. WHERE
9. { ?x rdf:type foaf:Person .
10. ?x foaf:member/foaf:page "www.pucrio.br" }
```

Note that  $F_R$  must be executed before  $u$  is. Assume that  $F_R$  returns just two IRIs:

$\mathbf{R} = \{ \langle \text{http://example/re1} \rangle, \langle \text{http://example/re2} \rangle \}$

**Step 2.** We proceed by deleting from  $L$  all sameAs links of the form  $(s, owl:sameAs, o)$ , where  $s \in \mathbf{R}$ . This is expressed as the following SPARQL deletion (assume that  $L$  is stored in  $\langle \text{http://sameRe.org} \rangle$ ):

```
11. WITH <http://sameRe.org>
12. DELETE WHERE{ { <http://example/re1> owl:sameAs ?x } UNION
13. { <http://example/re2> owl:sameAs ?x } }
```

To retrieve the properties of  $\langle \text{http://example/re1} \rangle$ , we use a query  $F_{rel}$ , synthesized from  $F_{SWCC}$ :

```
14. CONSTRUCT {<http://example/re1> rdf:type foaf:Person .
15. <http://example/re1> foaf:firstName ?fn .
16. <http://example/re1> foaf:lastName ?ln .
17. <http://example/re1> foaf:workplaceHomepage ?op }
18. WHERE
19. { GRAPH <http://data.semanticweb.org>
20. { <http://example/re1> rdf:type foaf:Person .
21. <http://example/re1> foaf:firstName ?fn .
22. <http://example/re1> foaf:lastName ?ln .
23. <http://example/re1> foaf:member/foaf:page ?op } }
```

To retrieve the properties of  $\langle \text{http://example/re2} \rangle$ , we use a query  $F_{re2}$ , likewise synthesized from  $F_{SWCC}$ . Note that, since the matching process has to access triples stored in  $BrCV$  through view  $ReBrCV$ , if insertions into  $SWCC$  are frequent, then it would be useful to explicitly materialize  $ReBrCV$ . Otherwise, it would be advantageous to maintain  $ReBrCV$  as a virtual set of triples.

We stress that the possibility of appropriating solutions proposed for materialized RDF view maintenance was indeed one of the major motivations for introducing views to address materialized sameAs linkset maintenance. Furthermore, we note that the use of views defined by simple property path queries was a pragmatic decision since the matching tools adopt property paths to specify the values to be matched.

Finally, we suggest two types of components, called *view controllers* and *linkset controllers*, to include in triple stores to improve the support for dataset interlinking (see Figure 1). The *view controller* for a view  $v$  over a dataset  $T$ , has the following functionality:

- Accept registrations from *linkset controllers* that will consume data through  $v$ .
- Offer a SPARQL endpoint to access  $v$  (useful to create a linkset using  $v$ ).
- Monitor each update on  $T$  that affects  $v$  and create a set  $R$  of IRIs (as in Step 1 discussed in this section).
- Send  $R$  to the *linkset controllers* registered with itself (useful to maintain a linkset created using  $v$ ); the *view controller* may send the sets of IRIs one-by-one, or in batch, starting from a given timestamp.

The *linkset controller* for a linkset  $L$ , defined over views  $v$  and  $w$ , has the following functionality:

- Register itself with the *view controllers* for  $v$  and  $w$ .
- Initialize  $L$  by accessing  $v$  and  $w$ .
- Receive (or request) sets of IRIs from the *view controllers* for  $v$  and  $w$  and update  $L$  accordingly (as in Step 2 discussed in this section); the *linkset controller* may receive the sets one-by-one, or request them in batch.

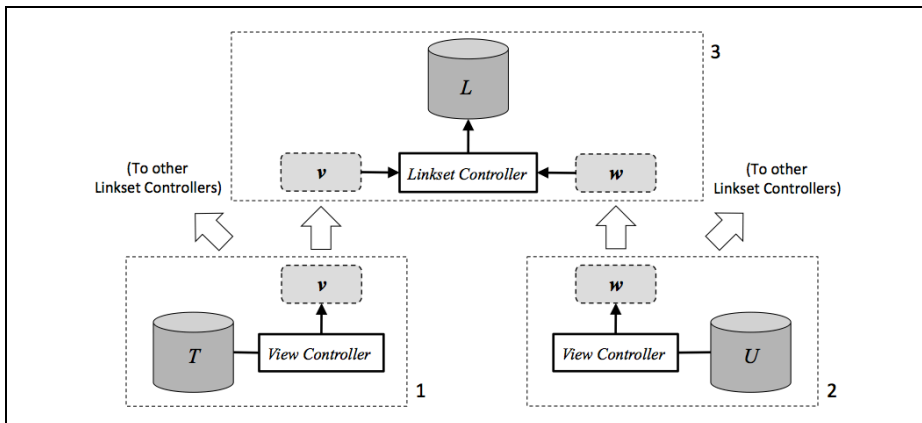


Fig. 1. Suggested architecture to create and maintain materialized sameAs linksets.

## 5 Conclusions

In this paper, we analyzed the major issues that dataset interlinking, using materialized sameAs linksets, raises. We introduced views to simplify the creation of sameAs linksets and showed that materialized sameAs linkset maintenance can appropriate the solutions designed for materialized view maintenance. Then, we proposed two types of components that triple stores should include to improve the support for materialized sameAs linksets, both at the creation and the maintenance stages.

The discussion in this paper benefited from early implementations of a sameAs linkset maintenance tool [9]. As for future plans, we will work with more general classes of views to create and maintain materialized (generic) linksets.

**Acknowledgments.** This work was partly supported by CNPq, under grants 160326/2012-5, 303332/2013-1 and 57128/2009-9, and by FAPERJ, under grants E-26/170028/2008 and E-26/103.070/2011.

## References

1. Alexander, K., Cyganiak, R., Hausenblas, M., Zhao, J. (2011) Describing Linked Triplesets with the VoID Vocabulary. W3C Interest Group Note 03 March 2011.
2. Berners-Lee, T. (2006) Linked Data, <http://www.w3.org/DesignIssues/LinkedData.html>.
3. Isele, R., Jentzsch, A., Bizer, C. (2011) Efficient Multidimensional Blocking for Link Discovery without losing Recall. Proc. WebDB 2011.
4. Leme, L.A.P.P., Lopes, G.R., Nunes, B.P., Casanova, M.A., Dietze, S. (2013) Identifying candidate triplesets for data interlinking. Proc. ICWE 2013, pp. 354-366.
5. Lopes, G.R., Leme, L.A.P.P., Nunes, B.P., Casanova, M.A., Dietze, S. (2013) Recommending Triplet Interlinking through a Social Network Approach. Proc. WISE 2013, pp. 149-161.
6. Möller, K., Bechhofer, S., Heath, T. (2009) Semantic Web Conference Ontology, <http://data.semanticweb.org/ns/swc/ontology>.
7. Nikolov, A., d'Aquin, M., Motta, E. (2012) What Should I Link to? Identifying Relevant Sources and Classes for Data Linking. In: JIST'12, Springer (2012), pp. 284-299.
8. Ngonga Ngomo, A.-C., Auer, S. (2011) LIMES - A Time-Efficient Approach for Large-Scale Link Discovery on the Web of Data. Proc. IJCAI 2011, pp. 2312-2317.
9. Ourofino, C.G. (2013). *Materialização e Manutenção de Ligações owl:sameAs*. M.Sc. Dissertation. Dept. Informatics, Pontifical Catholic University of Rio de Janeiro.
10. Popitsch, N., Haslhofer, B. (2011) DSNotify – A Solution for event detection and link maintenance in dynamic triplesets. Journal of Web Semantics, 9(3), pp. 266-283.
11. Staudt, M., Jarke, M. (1996) Incremental maintenance of externally materialized views. Proc. VLDB 1996, pp. 75-86.
12. Vidal, V.M.P., Casanova, M.A., Cardoso, D.S. (2013) Incremental Maintenance of RDF Views of Relational Data. Proc. ODBASE 2013, pp 572-587.
13. Volz, J., Bizer, C., Gaedke, M., Kobilarov, G. (2009) Discovering and Maintaining Links on the Web of Data. Proc. ISWC 2009, pp. 650-665.