

Using Changesets for Incremental Maintenance of Linkset Views

Vânia M. P. Vidal¹, Marco A. Casanova², Elisa S. Menendez², Narciso Arruda¹,
Valeria M. Pequeno³, Luiz A. Paes Leme⁴

¹Federal University of Ceará, Fortaleza, CE, Brazil
{vvidal, narciso}@lia.ufc.br

²Department of Informatics – Pontifical Catholic University of Rio de Janeiro, RJ, Brazil
casanova@inf.puc-rio.br, elisamenendez@gmail.com

³INESC-ID – Porto Salvo - Portugal
vmp@inesc-id.pt

⁴Fluminense Federal University, Niteroi, RJ, Brazil
lapaesleme@ic.uff.br

Abstract. In the Linked Data field, data publishers frequently materialize linksets between two datasets using link discovery tools. However, when the datasets are continually updated, a materialized linkset must also be updated since the links may no longer meet the linkage rules. To help solve this problem, this paper presents an approach for maintaining linksets, which treats linksets as materialized views, is based on changesets and adopts an incremental strategy. The paper formalizes the materialized linkset maintenance problem based on changesets and indicates that our approach correctly maintains materialized linksets views. Finally, it suggests an architecture and describes an implementation and experiments to validate the proposed approach.

Keywords: RDF dataset interlinking, Linked Data, View Maintenance.

1 Introduction

The Linked Data initiative [1] defines best practices for publishing and interlinking data on the Web using RDF triples to represent the data. Briefly, a *dataset* is simply a set of RDF triples. A *link* is an RDF triple of the form (s, p, o) , where s and o are resources defined in two distinct datasets. A *linkset* is a set of links.

Link discovery tools help create and materialize linksets. These tools are typically semi-automatic in the sense that users have to define a set of *linkage rules* that specify conditions that resources must fulfill to be interlinked. However, when datasets are continually updated, the maintenance of a materialized linkset requires attention since the links may no longer meet the linkage rules that originated the linkset. To inform consumers about changes, an RDF dataset should publish *changesets* [3] to indicate the difference between two states of the dataset.

In this paper, we present an approach for maintaining materialized linksets. The approach we propose: (1) treats linksets as materialized views, called *linkset views*; (2) accounts for the facts that a linkset is computed by (complex) linkage rules and

that the linkset does not contain the property values used by the linkage rules; (3) uses the changesets published by the source datasets to compute the changes that must be applied to a materialized linkset to keep it consistent with the new states of the source datasets; (4) adopts an incremental strategy. The proposed approach has two main steps. The first step uses the changesets, published by the source datasets, to compute the set of updated resources that are relevant to the materialized linkset. The second step updates the links for the relevant resources.

The contributions of the paper are: (i) we formalize the materialized linkset maintenance problem based on changesets; (ii) we define an approach that uses changesets to incrementally maintain materialized linksets and informally illustrate how it works; (iii) we provide two theorems that indicate that the proposed algorithms correctly maintains materialized linksets views; (iv) we describe an implementation and experiments to validate the approach.

Several tools were designed to create linksets [4][5][9]. The introduction of views, as suggested in [2], would simplify the configuration of the tools designed to create links. In another direction, tools, such as DSNotify [6], were designed to inform database administrators about dataset changes and to allow them to preserve link integrity. The proposed approach is based on, but not reducible to such incremental view maintenance strategies. Indeed, a linkset is not a regular view computed by querying two datasets, but it is created using linkage rules that frequently involve computing entity similarity. Furthermore, a linkset does not contain the property values that the linkage rules use. Endris et al. [3] presented a framework for interest-based RDF update propagation that can consistently maintain a full or partial replication of large LOD datasets. This framework is also based on changesets, but the solution can only be applied when the view mappings are direct mappings. The approach proposed in this paper goes further and considers linkset views defined by expressive mappings.

The paper is organized as follows. Section 2 introduces basic definitions and a running example. Section 3 presents our approach for maintaining linksets views. Section 4 describes an implementation and experiments to validate the proposed approach. Finally, Section 5 contains the conclusions.

2 Linkset Views

2.1 Linkset View Definition

To make the paper self-contained, we introduce an abstract notation to define *catalogue views* and *linkset views* with the help of mapping rules [7]. In the rest of this paper, $\sigma_S(t)$ denotes the state of S in time t , where S can be a source dataset or a view, and $M[\sigma_S(t)]$ denotes the set of triples defined by a set M of mapping rules against $\sigma_S(t)$.

A *catalogue view definition* is a triple $\mathbf{V} = (V_V, S_V, M_V)$, where

- V_V is the vocabulary of \mathbf{V} , also called the *view vocabulary*, and consists of a single class and a set of datatype properties

- S_V is the source dataset which exports the view \mathbf{V} , described by a vocabulary V_S
- M_V is a set of mapping rules that map concepts of V_S to concepts of V_V , called the *view mapping*.

A *materialization* of view \mathbf{V} at time t is obtained by computing $M_V[\sigma_{S_V}(t)]$ and storing it as part of a dataset.

A *linkset view definition* is a quintuple $\mathbf{L} = (P, V_L, \mathbf{F}, \mathbf{G}, \mu)$, where

- P is an object property
- V_L is the *match vocabulary* of \mathbf{L} and consists of a single class and a set of datatype properties
- $\mathbf{F} = (V_F, S_F, M_F)$ and $\mathbf{G} = (V_G, S_G, M_G)$ are catalogue view definitions where $V_F = V_G = V_L$. Thus, V_L is the common vocabulary for exported views \mathbf{F} and \mathbf{G}
- μ is a $2n$ -relation, called the *match predicate* of \mathbf{L} .

Let $V_L = \{C, P_1, \dots, P_n\}$ be the match vocabulary of \mathbf{L} . Let $\sigma_F(t)$ and $\sigma_G(t)$ be states respectively of \mathbf{F} and \mathbf{G} in time t . The *state* of \mathbf{L} in time t is the set $\sigma_L(t)$ defined as:
 $(s, p, o) \in \sigma_L(t)$ iff there are triples $(s, \text{rdf:type}, C)$, $(s, P_1, s_1), \dots, (s, P_n, s_n) \in \sigma_F(t)$ and $(o, \text{rdf:type}, C)$, $(o, P_1, o_1), \dots, (o, P_n, o_n) \in \sigma_G(t)$ such that $(s_1, \dots, s_n, o_1, \dots, o_n) \in \mu$

2.2 Running Example

In this section, we illustrate how to define a linkset view. Consider the *MusicBrainz* (<http://musicbrainz.org/doc/about>) dataset, which uses the Music ontology. Figure 1 shows a fragment of the Music ontology, which reuses terms from three well-known vocabularies: *FOAF* (Friend of a Friend), *MO* (Music Ontology) and *DC* (Dublin Core). Consider the *DBpedia* (<http://wiki.dbpedia.org/about>) dataset, which uses the *DBpedia* Ontology (*dbo*). Figure 2 shows a fragment of *DBpedia* ontology.

Suppose that a user wants to create *sameAs* links between instances of the class *Record* in the *MusicBrainz* dataset and instances of the class *Album* in the *DBpedia* dataset. For this purpose, the user creates the linkset view definition $\mathbf{L} = (\text{owl:sameAs}, V_L, \mathbf{F}, \mathbf{G}, \mu)$, where: $V_L = \{\text{mo:Record}, \text{dc:title}, \text{mvl:artistName}, \text{dbo:releaseDate}\}$; the vocabulary V_L reuses terms from *DBpedia* and *Music Ontologies* and defines a new term *mvl:artistName*; \mathbf{F} and \mathbf{G} are catalogues views exported by *DBpedia* and *MusicBrainz*, respectively, with mapping rules M_F and M_G from *DBpedia* and *Music Ontology* to common vocabulary V_L , respectively:

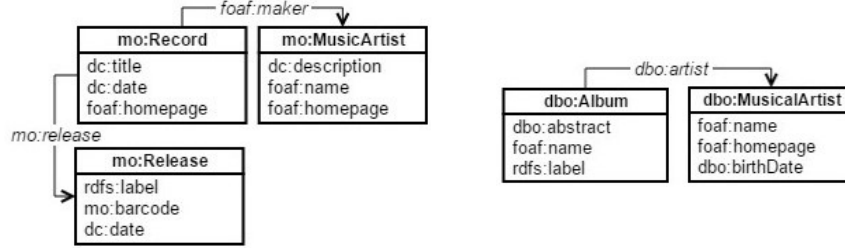
M_F : $\text{mo:Record}(x) \leftarrow \text{dbo:Album}(x)$
 $\text{dc:title}(x, y) \leftarrow \text{dbo:Album}(x); \text{foaf:name}(x, y)$
 $\text{mvl:artistName}(x, y) \leftarrow \text{dbo:Album}(x); \text{dbo:artist}(x, z); \text{foaf:name}(z, y)$
 $\text{dbo:releaseDate}(x, y) \leftarrow \text{dbo:Album}(x); \text{dbo:releaseDate}(x, y).$

M_G : is omitted here due to space limitation

and μ is the match predicate defined as

$(s_1, s_2, s_3, o_1, o_2, o_3) \in \mu$ iff $\sigma(s_k, o_k) \geq \alpha$, for each $k=1,2,3$

where σ is the 3-gram distance and $\alpha = 0.5$. The match predicate compares the *title*, *artistName* and *releaseDate* of instances of *Record* from both views \mathbf{F} and \mathbf{G} .



3 Linkset Incremental Maintenance Based on Changesets

In this section, we present our approach to correctly compute the changeset for a linkset view L , based on the changesets published by S_F and S_G . A *changeset* of an RDF dataset S from the state $\sigma_S(t_0)$ in time t_0 to the state $\sigma_S(t_1)$ in time t_1 is a pair $\langle \Delta_S^-(t_0, t_1), \Delta_S^+(t_0, t_1) \rangle$, where $\Delta_S^-(t_0, t_1)$ is the set of triples removed from $\sigma_S(t_0)$ and $\Delta_S^+(t_0, t_1)$ is the set of triples added to $\sigma_S(t_0)$ to create $\sigma_S(t_1)$ (for a formal definition see [8]). The approach we suggest to compute $\Delta_L(t_0, t_1)$ follows two main steps: (1) Compute $R_F(t_0, t_1)$, the set of resources that are affected by $\Delta_{S_F}(t_0, t_1)$ w.r.t F , and $R_G(t_0, t_1)$, the set of resources that are affected by $\Delta_{S_G}(t_0, t_1)$ w.r.t G ; (2) Compute $\Delta_L(t_0, t_1)$ using the resources in $R_F(t_0, t_1)$ and $R_G(t_0, t_1)$.

3.1 Computing the Affected Resources

In this section we present an algorithm to compute $R_F(t_0, t_1)$, the set of resources that are affected by $\Delta_{S_F}(t_0, t_1)$ w.r.t F . We say that a resource s is *affected* by $\Delta_{S_F}(t_0, t_1)$ iff the state of s in $\sigma_F(t_0)$ is different from the state of s in $\sigma_F(t_1)$. More formally, a resource s is *affected* by $\Delta_{S_F}(t_0, t_1)$ w.r.t F iff $s[\sigma_F(t_0)] \neq s[\sigma_F(t_1)]$. To compute $R_F(t_0, t_1)$, we have to consider two situations:

- (i) If all mappings in M_F are simple mappings, $R_F(t_0, t_1)$ can be directly computed from $\Delta_{S_F}(t_0, t_1)$ [8].
- (ii) Otherwise, the computation of $R_F(t_0, t_1)$ requires, besides $\Delta_{S_F}(t_0, t_1)$, the old state $\sigma_{S_F}(t_0)$ of S_F . But, $\sigma_{S_F}(t_0)$ is no longer available when the changeset is published.

To account for the second case, we introduce the notion of *auxiliary view* A_F for F , defined as a triple $A_F = (V_{A_F}, S_{A_F}, M_{A_F})$, where:

- V_{A_F} consists of all classes and properties in V_F that are relevant to S_F
- $S_{A_F} = S_F$
- M_{A_F} is a set of direct mappings from the vocabulary of S_F to V_{A_F} .

In the suggested architecture (see [8]), the auxiliary view A_F is materialized, while the view F is virtual. Algorithm 1, shown in Table 1, computes $R_F(t_0, t_1)$ when an auxiliary view is required. Theorem 1 in [8] shows that Algorithm 1 correctly computes the set of affected resources.

Table 1: Algorithm 1

<p>Input: $\sigma_{\mathbf{AF}}(t_0), \Delta_{\mathbf{SF}}(t_0, t_1)$</p> <p>Step 1.1: Compute $\Delta_{\mathbf{AF}}^-(t_0, t_1) = M_{\mathbf{AF}}[\Delta_{\mathbf{SF}}^-(t_0, t_1)]$ and $\Delta_{\mathbf{AF}}^+(t_0, t_1) = M_{\mathbf{AF}}[\Delta_{\mathbf{SF}}^+(t_0, t_1)]$;</p> <p>Step 1.2: Compute $R^-(t_0, t_1) = \{s \mid s \text{ is the subject of a triple } t \text{ in } \sigma_{\mathbf{F}}(t_0) \text{ and } t \text{ is affected by a triple in } \Delta_{\mathbf{AF}}^-(t_0, t_1)\}$;</p> <p>Step 1.3: Compute $\sigma_{\mathbf{AF}}(t_1) = (\sigma_{\mathbf{AF}}(t_0) - \Delta_{\mathbf{AF}}^-(t_0, t_1)) \cup \Delta_{\mathbf{AF}}^+(t_0, t_1)$;</p> <p>Step 1.4: Compute $R^+(t_0, t_1) = \{s \mid s \text{ is the subject of a triple } t \text{ in } \sigma_{\mathbf{F}}(t_1) \text{ and } t \text{ is affected by a triple in } \Delta_{\mathbf{AF}}^+(t_0, t_1)\}$;</p> <p>Step 1.5: Return $R_{\mathbf{F}}(t_0, t_1) = R^-(t_0, t_1) \cup R^+(t_0, t_1)$.</p>
--

To illustrate the computation of $R_{\mathbf{F}}(t_0, t_1)$ by Algorithm 1, consider the linkset view \mathbf{L} over the catalogue views \mathbf{F} and \mathbf{G} exported from *DBpedia* and *MusicBrainz*, defined in Section 2.2. The auxiliary view for \mathbf{F} is $\mathbf{AF} = (V_{\mathbf{AF}}, S_{\mathbf{AF}}, M_{\mathbf{AF}})$, where: $V_{\mathbf{AF}} = \{dbo:Album, foaf:name, dbo:artist, dbo:releaseDate\}$; $S_{\mathbf{AF}}: \text{http://host/dbpedia}$; $M_{\mathbf{AF}}$ is a set of direct mappings from *DBpedia*'s vocabulary to $V_{\mathbf{AF}}$. The auxiliary view for \mathbf{G} is $\mathbf{AG} = (V_{\mathbf{AG}}, S_{\mathbf{AG}}, M_{\mathbf{AG}})$, where: $V_{\mathbf{AG}} = \{mo:Record, dc:title, foaf:maker, foaf:name, mo:realese, dc:date\}$; $S_{\mathbf{AG}}: \text{http://host/MusicBrainz}$; $M_{\mathbf{AG}}$ is a set of direct mappings from *MusicBrainz*'s vocabulary to $V_{\mathbf{AG}}$. Assume that:

- Table 2 shows the states of the catalogue views \mathbf{F} and \mathbf{G} , the auxiliary view \mathbf{AF} and linkset view \mathbf{L} , on Sep 10, 2015 at 10:00 AM (t_0) and the triples published by the DBpedia Live extractor for the changes made on Sep 10, 2015 between 10:00 AM (t_0) and 11:02 PM (t_1).
- *MusicBrainz* did not release new changeset on Sep 10, 2015 between 10:00 AM (t_0) and 11:02 PM (t_1).

Algorithm 1 computes the set $R_{\mathbf{F}}(t_0, t_1)$ in 5 steps:

Step 1.1: Compute $\Delta_{\mathbf{AF}}^-(t_0, t_1)$ and $\Delta_{\mathbf{AF}}^+(t_0, t_1)$. From Algorithm 1, we have:

$$\Delta_{\mathbf{AF}}^-(t_0, t_1) = \{(dbr:b1 \text{ foaf:name "Jackson Michael"})\}.$$

$$\Delta_{\mathbf{AF}}^+(t_0, t_1) = \{(dbr:a1 \text{ dbo:releaseDate "1982-11-29"}), (dbr:a1 \text{ foaf:name "Thriller"}), (dbr:b1 \text{ foaf:name "Michael Joseph Jackson"})\}.$$

Step 1.2: Compute $R^-(t_0, t_1)$. First we have to compute which triples in $\sigma_{\mathbf{F}}(t_0)$ are affected by triples in $\Delta_{\mathbf{AF}}^-(t_0, t_1)$. For example, consider the triple $\mathbf{y} = (dbr:b1 \text{ foaf:name "Jackson Michael"})$ in $\Delta_{\mathbf{AF}}^-(t_0, t_1)$. The triples $(dbr:a1 \text{ mvl:artistName "Jackson Michael"})$ and $(dbr:a2 \text{ mvl:artistName "Jackson Michael"})$ in $\sigma_{\mathbf{F}}(t_0)$ are affected by \mathbf{y} because those triples are generated by substituting $\text{foaf:name}(z, \mathbf{y})$ by \mathbf{y} in the mapping rule " $\text{mvl:artistName}(x, \mathbf{y}) \leftarrow \text{dbo:Album}(x); \text{dbo:artist}(x, z); \text{foaf:name}(z, \mathbf{y})$ ". Therefore, $R^-(t_0, t_1) = \{dbr:a1, dbr:a2\}$.

Step 1.3: Compute $\sigma_{\mathbf{AF}}(t_1) = (\sigma_{\mathbf{AF}}(t_0) - \Delta_{\mathbf{AF}}^-(t_0, t_1)) \cup \Delta_{\mathbf{AF}}^+(t_0, t_1)$ (See Table 2).

Step 1.4: Compute $R^+(t_0, t_1)$. First we have to compute the triples in $\sigma_{\mathbf{F}}(t_1)$ that are affected by triples in $\Delta_{\mathbf{AF}}^+(t_0, t_1)$. The triples $(dbr:a1 \text{ dbo:releaseDate "1982-11-29"}), (dbr:a1 \text{ dc:title "Thriller"}),$ and $(dbr:a1 \text{ mvl:artistName "Jackson Joseph Michael"})$ in $\sigma_{\mathbf{F}}(t_1)$ are affected by triples in $\Delta_{\mathbf{AF}}^+(t_0, t_1)$. Therefore, $R^+(t_0, t_1) = \{dbr:a1, dbr:a2\}$.

Step 1.5: Compute $R_{\mathbf{F}}(t_0, t_1) = R^-(t_0, t_1) \cup R^+(t_0, t_1)$

$$R_{\mathbf{F}}(t_0, t_1) = \{dbr:a1, dbr:a2\}.$$

Table 2: $\sigma_F(t_0)$, $\sigma_G(t_0)$, $\sigma_{AF}(t_0)$, $\sigma_{AF}(t_1)$, $\sigma_L(t_0)$, $\Delta_{DBpedia}^-(t_0, t_1)$ and $\Delta_{DBpedia}^+(t_0, t_1)$

$\sigma_F(t_0) = \{$ (dbr:a1 rdf:type <i>mo:Record</i>); (dbr:a1 mvl:artistName "Jackson Michael"); (dbr:a2 rdf:type <i>mo:Record</i>); (dbr:a2 dc:title "Thriller 25"); (dbr:a2 mvl:artistName "Jackson Michael"); (dbr:a2 dbo:releaseDate "2008-02-08") $\}$ $\sigma_{AF}(t_0) = \{$ (dbr:a1 rdf:type <i>dbo:Album</i>); (dbr:a1 dbo:artist dbr:b1); (dbr:a2 rdf:type <i>dbo:Album</i>); (dbr:a2 foaf:name "Thriller 25"); (dbr:a2 dbo:releaseDate "2008-02-08"); (dbr:a2 dbo:artist dbr:b1); (dbr:b1 foaf:name "Jackson Michael") $\}$ $\sigma_{AF}(t_1) = \{$ (dbr:a1 rdf:type <i>dbo:Album</i>), (dbr:a1 foaf:name "Thriller"), (dbr:a1 dbo:releaseDate "1982-11-29"), (dbr:a2 rdf:type <i>dbo:Album</i>), (dbr:a2 foaf:name "Thriller 25"), (dbr:a2 dbo:releaseDate "2008-02-08"), (dbr:a1 dbo:artist dbr:b1), $\}$	$\sigma_G(t_0) = \{$ (dbr:a2 dbo:artist dbr:b1), (dbr:b1 foaf:name "Michael Joseph Jackson") $\}$ $\sigma_L(t_0) = \{$ (mbr:r1 rdf:type <i>mo:Record</i>); (mbr:r1 dc:title "Thriller"); (mbr:r1 mvl:artistName "Michael Joseph Jackson"); (mbr:r1 dc:releaseDate "1982-11-29"); (mbr:r2 rdf:type <i>mo:Record</i>); (mbr:r2 dc:title "Thriller 25"); (mbr:r2 mvl:artistName "Michael Joseph Jackson"); (mbr:r2 dc:releaseDate "2008-02-08") $\}$ $\sigma_L(t_1) = \{(dbr:a2 owl:sameAs mbr:r2)\}$ $\Delta_{DBpedia}^-(t_0, t_1) = \{$ (dbr:b1 foaf:name "Jackson Michael") $\}$ $\Delta_{DBpedia}^+(t_0, t_1) = \{$ (dbr:a1 dbo:releaseDate "1982-11-29"); (dbr:a1 foaf:name "Thriller"); (dbr:b1 foaf:name "Michael Joseph Jackson") $\}$
--	---

3.2 Computing the Changeset for **L**

Algorithm 2 in Table 3 returns $\Delta_L(t_0, t_1)$, a changeset for **L**. Theorem 2 in [8] shows that the changeset $\Delta_L(t_0, t_1)$ returned by Algorithm 2 correctly maintains **L**. To illustrate the computation of $\Delta_L(t_0, t_1)$ by Algorithm 2, consider the set $R_F(t_0, t_1)$ computed in Section 3.1. Since we assume that *MusicBrainz* did not release new changesets in the time interval considered, $R_G(t_0, t_1) = \emptyset$. Algorithm 2 computes $\Delta_L(t_0, t_1)$ in 5 steps.

Step 2.1: Compute D_F and I_F . From Algorithm 2, we have:

$$D_F = \{(dbr:a2 owl:sameAs mbr:r2)\};$$

$$I_F = \{(dbr:a1 owl:sameAs mbr:r1), (dbr:a2 owl:sameAs mbr:r2)\}.$$

Step 2.2: Compute D_G and I_G . From Algorithm 2, we have:

$$D_G = \emptyset; I_G = \emptyset.$$

Step 2.3: Compute $\Delta_L^-(t_0, t_1)$. From Algorithm 2, we have:

$$\Delta_L^-(t_0, t_1) = \{(dbr:a2 owl:sameAs mbr:r2)\}.$$

Step 2.2: Compute $\Delta_L^+(t_0, t_1)$. From Algorithm 2, we have:

$$\Delta_L^+(t_0, t_1) = \{(dbr:a1 owl:sameAs mbr:r1), (dbr:a2 owl:sameAs mbr:r2)\}.$$

Step 2.5: Return $\Delta_L(t_0, t_1) = \langle \Delta_L^-(t_0, t_1), \Delta_L^+(t_0, t_1) \rangle$

The new state of **L** is computed by $\sigma_L(t_1) = (\sigma_L(t_0) - \Delta_L^-(t_0, t_1)) \cup \Delta_L^+(t_0, t_1)$. Therefore, $\sigma_L(t_1) = \{(dbr:a1 owl:sameAs mbr:r1), (dbr:a2 owl:sameAs mbr:r2)\}$.

Table 3: Algorithm 2

Input: $\sigma_{\mathbf{L}}(t_0), \sigma_{\mathbf{AF}}(t_1), \sigma_{\mathbf{AG}}(t_1), R_{\mathbf{F}}(t_0, t_1), R_{\mathbf{G}}(t_0, t_1)$
Step 2.1: Compute
$\mathbf{D}_{\mathbf{F}} = \{ (s, p, o) / (s, p, o) \in \sigma_{\mathbf{L}}(t_0) \text{ and } s \in R_{\mathbf{F}}(t_0, t_1) \}$
$\mathbf{I}_{\mathbf{F}} = \{ (s, p, o) / s \in R_{\mathbf{F}}(t_0, t_1), o \in M_{\mathbf{F}}[\sigma_{\mathbf{AF}}(t_1)] \}$
Step 2.2: Compute
$\mathbf{D}_{\mathbf{G}} = \{ (s, p, o) / (s, p, o) \in \sigma_{\mathbf{L}}(t_0) \text{ and } s \in R_{\mathbf{G}}(t_0, t_1) \}$
$\mathbf{I}_{\mathbf{G}} = \{ (s, p, o) / s \in R_{\mathbf{G}}(t_0, t_1), o \in M_{\mathbf{G}}[\sigma_{\mathbf{AG}}(t_1)] \}$
Step 2.3: Compute $\Delta_{\mathbf{L}}^{-}(t_0, t_1) = \mathbf{D}_{\mathbf{F}} \cup \mathbf{D}_{\mathbf{G}}$;
Step 2.4: Compute $\Delta_{\mathbf{L}}^{+}(t_0, t_1) = \mathbf{I}_{\mathbf{F}} \cup \mathbf{I}_{\mathbf{G}}$;
Step 2.5: Return $\Delta_{\mathbf{L}}(t_0, t_1) = \langle \Delta_{\mathbf{L}}^{-}(t_0, t_1), \Delta_{\mathbf{L}}^{+}(t_0, t_1) \rangle$.

4 Implementation and Experiments

The *Linkset Maintainer* tool was developed using Java, JBoss 7, Open Link Virtuoso as the triple store, and Silk as the link discovery tool. In order to evaluate the performance of the incremental strategy, we selected two datasets: a *Music Brainz* dump, and the *DBpedia* endpoint. Additionally, *DBpedia* daily provides sets of changed triples extracted from Wikipedia, called *DBpedia* Changesets (available at <http://live.dbpedia.org/changesets/>), which are organized by year, month, day, and hour; and also separated by the type of update (added, removed, reinserted, and clear).

We defined views about music records released after 2010 for each dataset. The “MusicBrainz_Records” view had 311,374 resources and the “DBpedia_Records” view had 35,651 resources. Then, we materialized an *owl:sameAs* linkset of records, using these views, by comparing their titles, artist names and release dates. The linkset had 14,716 links and the runtime to compute it using Silk was around 4 hours. To test the performance of the incremental strategy, we processed and analyzed one entire day (October 3th, 2015) of *DBpedia* changesets. We computed the total number of inserted and deleted resources, the sets $\Delta_{\mathbf{AF}}^{-}(t_0, t_1)$, $\Delta_{\mathbf{AF}}^{+}(t_0, t_1)$, R^{-} and R^{+} , and the runtime to maintain the linkset. Table 4 summarizes the results for the whole day.

Table 4: Analysis of *DBpedia* Changesets.

	Total	Sets	Avg	Max
Deleted Resources	144385	720	200,5	1230
Auxiliary View Deleted Resources ($\Delta_{\mathbf{AF}}^{-}(t_0, t_1)$)	7453	720	10,35	85
View Deleted Resources (R^{-})	318	720	0,4	16
Inserted Resources	134887	720	187,3	1072
Auxiliary View Inserted Resources ($\Delta_{\mathbf{AF}}^{+}(t_0, t_1)$)	7614	720	10,58	85
View Inserted Resources (R^{+})	372	720	0,5	16
Runtime	12 h	720	1 min	5 min

Note that, on the average, there are 200,5 deleted resources per changeset, of which only an average of 0,4 resources affected the view. Also note that the max number of R^{-} and R^{+} was only 85. We highlight that the max runtime to process a changeset was

5 minutes, which included the time to download and decompress the changeset file, compute R^- and R^+ , and update the linkset. Recall that the runtime to materialize the linkset was 4 hours, which is much higher than the max runtime to incrementally maintain the linkset using a changeset. Therefore, in these experiments, we showed that the incremental strategy, by far, outperformed the re-materialization strategy.

5 Conclusions

Data publishers frequently materialize linkset views between two source datasets using link discovery tools. However, when the source datasets are updated, the materialized linkset views must also be updated. To help solve this problem, we presented a formal framework for maintaining linkset views that adopts changesets and an incremental strategy. The changesets, published by the source datasets, are used to compute the set of updated resources that are relevant to a linkset view; the incremental strategy updates the links only for the relevant resources. We provided a formalization of our approach and indicated that the framework correctly maintains the linkset views. We also described experiments to validate the proposed framework.

Acknowledgments: This work was partly funded by CNPq under grants 153908/2015-7, 557128/2009-9, 444976/2014-0, 303332/2013-1, 442338/2014-7 and 248743/2013-9, by FAPERJ under grants e E-26-170028/2008 and E-26/201.337/2014 and by FCT – Fundação para a Ciência e Tecnologia, under grant SFRH/BPD/76024/2011.

References

1. Berners-Lee, T. (2006), <http://www.w3.org/DesignIssues/LinkedData.html>
2. Casanova, M. A., Vidal, V. M. P., Lopes, G. R., Leme, L. A. P. P., Ruback, L.: On Materialized sameAs Linksets. In: 25th International Conference on Database and Expert Systems Applications. pp. 377-384 (2014)
3. Endris, M. K., Faial, S., Orlandi, F., Auer, S., Scerri S.: Interest-based RDF Update Propagation. In: 14th International Semantic Web Conference. pp. 650-665 (2015)
4. Isele, R., Jentzsch, A., Bizer, C.: Efficient Multidimensional Blocking for Link Discovery without losing Recall. In: 14th International Workshop on the Web and Databases (2011)
5. Ngomo, A.C.N., Auer, S.: Limes: A time-efficient approach for large-scale link discovery on the web of data. In: 22nd International Joint Conference on Artificial Intelligence. pp. 2312–2317 (2011)
6. Popitsch, N., Haslhofer, B.: DSNotify – A Solution for event detection and link maintenance in dynamic triplesets. *Journal of Web Semantics*, 9(3), 266–283 (2011)
7. Vidal, V.M.P., et al.: Specification and Incremental Maintenance of Linked Data Mashup Views. In: 27th International Conference on Advanced Information Systems Engineering. pp. 214-229 (2015)
8. Vidal, V.M.P., Casanova, M.A., Menendez, E.S., Arruda, N.: A Formal Approach based on Changesets for the Incremental Maintenance of Linkset Views. Technical Report 015, Department of Informatics, PUC-Rio, Brazil (2015)
9. Volz, J., Bizer, C., Gaedke, M., Kobilarov, G.: Discovering and Maintaining Links on the Web of Data. In: 8th International Semantic Web Conference. pp. 650-665 (2009)