Co-Evolution of RDF Datasets

Sidra Faisal

Kemele M. Endris

Saeedeh Shekarpour
Wright State University - Main Campus, saeedeh.shekarpour@wright.edu

Sören Auer

Maria-Esther Vidal

Follow this and additional works at: https://corescholar.libraries.wright.edu/knoesis

Part of the Bioinformatics Commons, Communication Technology and New Media Commons, Databases and Information Systems Commons, OS and Networks Commons, and the Science and Technology Studies Commons

Repository Citation
Faisal, S., Endris, K. M., Shekarpour, S., Auer, S., & Vidal, M. (2016). Co-Evolution of RDF Datasets. Lecture Notes in Computer Science, 9671, 225-243.
https://corescholar.libraries.wright.edu/knoesis/1125

This Conference Proceeding is brought to you for free and open access by the The Ohio Center of Excellence in Knowledge-Enabled Computing (Kno.e.sis) at CORE Scholar. It has been accepted for inclusion in Kno.e.sis Publications by an authorized administrator of CORE Scholar. For more information, please contact library-corescholar@wright.edu.
Co-evolution of RDF Datasets

Sidra Faisal, Kemele M. Endris, Saeedeh Shekarpour, Sören Auer, Maria-Esther Vidal
University of Bonn & Fraunhofer IAIS, Bonn, Germany
(lastname)@cs.uni-bonn.de

Abstract. Linking Data initiatives have fostered the publication of large number of RDF datasets in the Linked Open Data (LOD) cloud, as well as the development of query processing infrastructures to access these data in a federated fashion. However, different experimental studies have shown that availability of LOD datasets cannot be always ensured, being RDF data replication required for envisioning reliable federated query frameworks. Albeit enhancing data availability, RDF data replication requires synchronization and conflict resolution when replicas and source datasets are allowed to change data over time, i.e., co-evolution management needs to be provided to ensure consistency. In this paper, we tackle the problem of RDF data co-evolution and devise an approach for conflict resolution during co-evolution of RDF datasets. Our proposed approach is property-oriented and allows for exploiting semantics about RDF properties during co-evolution management. The quality of our approach is empirically evaluated in different scenarios on the DBpedia-live dataset. Experimental results suggest that proposed proposed techniques have a positive impact on the quality of data in source datasets and replicas.

Key words: Dataset Synchronization, Dataset Co-evolution, Conflict Identification, Conflict Resolution, RDF Dataset

1 Introduction

During the last decade, the Linked Open Data (LOD) cloud has considerably grown [20], comprising currently more than 85 billion triples from approximately 3400 datasets. Further, Web based interfaces such as SPARQL endpoints [9] and Linked Data fragments [23], have been developed to access RDF data following the HTTP protocol, while federated query processing frameworks allow users to pose queries against federations of RDF datasets. Nevertheless, empirical studies by Buil-Aranda et al. [6] suggest the lack of Web availability of a large number of LOD datasets, being frequently required the replication of small portions of data, i.e., slices of an RDF dataset, to enhance reliability and performance of Linked Data applications [7]. Although RDF replication allows for enhancing RDF data availability, synchronization problems may be generated because source datasets

1 Observed on 17th December 2015 on http://stats.lod2.eu/
and replicas may change over time, e.g., *DBpedia Live mirror tool* publishes changes in a public changesets folder.

Co-evolution refers to mutual propagation of the changes between a replica and its origin or source dataset, where propagation specially in a mutual way, raises synchronization issues which need to be addressed to avoid data inconsistency. Issues are about how changes should be propagated and in case of inconsistencies or data conflicts, how these conflicts should be resolved. Thus, our main research problem is to develop a co-evolution process able to exploit the properties of RDF data and solve conflicts generated by the propagation of changes among source datasets and replicas. We propose a two-fold co-evolution approach, comprised of the following components: i) an RDF data synchronization component, and ii) a component for conflict identification and resolution.

Our approach relies on the assumption that either the source dataset provides a tool to compute a changeset at real-time or third party tools can be used for this purpose. Another assumption is that slices of the RDF data from the source dataset are replicated in the replicas or target datasets, where a slice corresponds to an RDF subgraph of the source RDF graph.

Figure 1 illustrates the co-evolution between two RDF datasets. Initially, a slice of the source dataset is used to create a target dataset, i.e., the target dataset \( T_{t_0} \) is sliced from the source dataset \( S_{t_0} \) of dataset \( S \) at time \( t_0 \). Both the source and target datasets evolve themselves with the passage of time, e.g., these datasets evolve to \( S_{t_i} \) and \( T_{t_j} \) during timeframe \( t_i - t_j \), while \( t_i < t_j \). Changes from \( S_{t_i} \), denoted by \( \delta(S_{t_i-t_j}) \), are propagated to the target and vice versa by the RDF data synchronization component. For synchronization, changes from both source and target datasets are compared to identify conflicts. The resolved conflicts are applied on the source and target datasets to vanish inconsistencies, for example, at time point \( t_j \), the co-evolution manager identifies the conflicts and resolves them. The conflicts are resolved and final changes are merged in both datasets.

We empirically evaluate the quality of our co-evolution approach on different co-evolution scenarios of data from the DBpedia and changesets from DBpedia-live published from September 01, 2015 to October 31, 2015 using iRap. The goal of the evaluation is to study the impact on data quality of the propose co-evolution process, where quality is measured in terms of completeness, consistency, and conciseness. Observed experimental results suggest that our synchronization, and conflict identification and resolution techniques positively affect the quality of the data in both the source and target datasets.

The paper is structured as follows: Section 3 provides formal definitions of the basic notations and concepts used in the proposed co-evolution approach. Section 4 presents detailed problem description and different synchronization strategies. We then present the proposed approach in Section 5 followed by

---

2 https://github.com/dbpedia/dbpedia-live-mirror
3 http://live.dbpedia.org/changesets/
4 An RDF slice is also known as a fragment in the approaches proposed by Ibañez et al. [10], Montoya et al. [15], and Verborgh et al. [23].
5 http://wiki.dbpedia.org/
evaluation in Section 6 presents the related work. We close with the conclusion and the directions for the future work.

2 Motivating example

Let us assume an application which requires information of politicians (e.g., name, birthYear, and spouse). This information can be sliced from the datasets like DBpedia\(^6\) and used locally by the application. We use the following SPARQL query to slice DBpedia for our use case scenario:

\[
\text{CONSTRUCT WHERE} \left\{ \begin{array}{l}
\text{?s rdf:type dbo:Politician.} \\
\text{OPTIONAL} \left\{ \begin{array}{l}
\text{?s foaf:name ?name.} \\
\text{?s dbp:birthYear ?birthYear.} \\
\text{?s dbp:spouse ?spouse.} \\
\text{?s owl:sameAs ?sameAs} \\
\end{array} \right. \\
\end{array} \right. 
\]

Our approach is inspired from the scenario described in Figure 2. Initially, at time \(t_0\), this slice is used to populate target dataset. Both source and target datasets evolve during timeframe \(t_i - t_j\), while \(t_i < t_j\). Source dataset adds object value \(dbo:Agent\) for rdf:type, \(AdrianSanders\) for foaf:name, 1959 for \(dbp:birthYear\), and \(Freebase:AdrianSanders\) and http://wikidata.org/entity/Q479047 for owl:sameAs to resource dbr:Adrian_Sanders. Target dataset adds object value \(dbo:MemberOfParliment\) for rdf:type, \(Sanders, Adrian\) for foaf:conname, and \(Freebase:AdrianSanders\) and http://yago-knowledge.org/resource/Adrian_Sanders\(^6\) for owl:sameAs to resource dbr:Adrian_Sanders.

\(^6\) http://dbpedia.org
Fig. 2: Motivating example: a) Target dataset initialization, b) evolution, and c) synchronization with source.

For resource dbr:Adrian_Sanders, we have two different values for rdf:type in source and target changesets. We need to check which of them is correct. We already know dbr:Adrian_Sanders can be an agent and member of parliament at the same time. However, this check can be made by looking whether the two classes are disjoint or not. Source adds object value 1959 for dbp:birthYear to dbr:Adrian_Sanders. As dbp:birthYear is a functional property, it can have only one value. So, we have to choose one value among the already existing value \(1959 - 01 - 01\) in dataset and the new value \(1959\) in the changeset. One solution can be to randomly select one value among two. Similarly, source adds object value Freebase : AdrianSanders for owl:sameAs while target dataset deletes this value after adding it. Considering target as a more customized dataset, we prefer the changes of target over source changes. Thus, we delete Freebase : AdrianSanders in synchronized dataset. We still have two different owl:sameAs values for dbr:Adrian_Sanders. However, as they are representing the same resource, we will keep both values in synchronized dataset.
3 Preliminaries

In this section, we formalize the main concepts required for realizing co-evolution of RDF datasets. The Resource Description Framework (RDF) is widely used to represent information on the Web. A resource can be anything (either physical or conceptual). The RDF data model expresses statements about Web resources in the form of subject-predicate-object (triple). The subject denotes a resource; the predicate expresses a property of subject or a relationship between the subject and the object; the object is either a resource or literal. For identifying resources, RDF uses Uniform Resource Identifiers (URIs) and Internationalized Resource Identifier (IRIs). The rationale behind is that the names of resources must be universally unique. We assume that both source and target datasets are RDF datasets. An RDF dataset is formally defined as follows:

**Definition 1 (RDF Dataset).** Formally, an RDF dataset is a finite set of triples \( (s, p, o) \in (I \cup B) \times I \times (I \cup L \cup B) \), where \( I, B, \) and \( L \) are the disjoint sets of all IRIs, blank nodes, and literals.

Let us assume that the slice contains the following triples:

| dbr:Adrian_Sanders rdf:type dbp:Politician; dbp:spouse Alison Sanders; dbp:birthYear 1959-01-01 (xsd:date). |

Listing 1.1: Content of initial target dataset

This local copy of sliced dataset, referred as target dataset, might undergo changes by user feedback (e.g., user can update the restaurant rating or fulfill abstract information). After some time, DBpedia dataset also evolves by adding new restaurants information or updating the existing ones. As a result, target dataset might be out of date and need to be synchronized with DBpedia. During synchronization, a conflict (defined in Definition 5) might occur, if the same information was updated by the source (DBpedia) dataset and the target dataset (by the app users).

**Definition 2 (Evolving RDF Dataset).** Let us assume that \( D_{t_i} \) represents the version of the RDF dataset \( D \) at the particular time \( t_i \). An evolving dataset \( D \) is a dataset whose triples change over time. In other words, for timeframe \( t_i - t_j \), there is a triple \( x \) such as either \( (x \in D_{t_i} \land x \notin D_{t_j}) \) or \( (x \notin D_{t_i} \land x \in D_{t_j}) \).

**Definition 3 (Changeset).** Let us assume that \( D \) is an evolving RDF dataset, and \( D_{t_i} \) is the version of \( D \) at time \( t_i \). A changeset which is denoted by \( \delta(D_{t_i}, t_j) \) shows the difference of two versions of an evolving RDF dataset in a particular timeframe \( t_i - t_j \), while \( t_i < t_j \). The changeset is formally defined as \( \delta(D_{t_i}, t_j) = \delta(D_{t_i}, t_j)^+ \cup \delta(D_{t_i}, t_j)^- \) where,

1. [http://www.w3.org/TR/rdf11-concepts/](http://www.w3.org/TR/rdf11-concepts/)
2. A URI is a string of characters used as unique identifier for a Web resource.
3. A generalization of URIs enabling the use of international character sets.
\(-\delta(D_{t_i-t_j})^+\) is a set of triples which have been added to the version \(D_{t_j}\) in comparison to the version \(D_{t_i}\).

\(-\delta(D_{t_i-t_j})^-\) is a set of triples which have been deleted from the version \(D_{t_j}\) in comparison to the version \(D_{t_i}\).

**Example 1 (Changesets).** Let the following files are found as changesets at time \(t_i\) from the source and target datasets.

| # (A). Deleted triples | # (B). Added triples |
|-------------------------|----------------------|
| dbr:Adrian_Sanders dbp:spouse Alison Sanders; | dbr:Adrian_Sanders rdf:type dbo:Agent; |
|   owl:sameAs Freebase:Adrian Sanders; |   foaf:name Adrian Sanders; |
|   dbp:birthYear 1959; |   db:p:MemberOfParliament; |
|   owl:sameAs http://wikidata.org/entity/Q479047. |   foaf:name Sanders, Adrian; |
|   |   owl:sameAs Freebase:Adrian Sanders; |
|   |   ow\(l\):sameAs http://yago-knowledge.org/resource/Adrian_Sanders. |

Listing 1.2: Source changeset, \((A)=\delta(S_{t_i-t_j})^-,\) and \((B)=\delta(S_{t_i-t_j})^+\)

| # (A). Deleted triples | # (B). Added triples |
|-------------------------|----------------------|
| dbr:Adrian_Sanders dbp:spouse Alison Sanders; | dbr:Adrian_Sanders rdf:type dbo:Agent; |
|   owl:sameAs Freebase:Adrian Sanders; |   foaf:name Adrian Sanders; |
|   dbp:birthYear 1959; |   db:p:MemberOfParliament; |
|   owl:sameAs http://wikidata.org/entity/Q479047. |   foaf:name Sanders, Adrian; |
|   |   owl:sameAs Freebase:Adrian Sanders; |
|   |   ow\(l\):sameAs http://yago-knowledge.org/resource/Adrian_Sanders. |

Listing 1.3: Target changeset, \((A)=\delta(T_{t_i-t_j})^-,\) and \((B)=\delta(T_{t_i-t_j})^+\)

**Definition 4 (Synchronized Dataset).** Two evolving datasets, \(D^{(1)}\) and \(D^{(2)}\), are said to be synchronized (or in sync) iff one of the following is true at a given time \(t_k\): i) \(D^{(1)}_{t_k} \subseteq D^{(2)}_{t_k}\), ii) \(D^{(2)}_{t_k} \subseteq D^{(1)}_{t_k}\), or iii) \(D^{(1)}_{t_k} \equiv D^{(2)}_{t_k}\).

4 Problem Statement

The core of the co-evolution concept relies on the mutual propagation of changes between the source and target datasets in order to keep the datasets in sync. Thus, from time to time, the target dataset and the source dataset have to exchange the changesets and then update the local repositories. Updating a dataset with changesets from the source dataset might cause inconsistencies. Our co-evolution strategy aims at dealing with changesets from either the source or target dataset and provide a suitable reconciliation strategy. Various strategies can be employed for synchronising datasets. In this section we provide requirements and formal definitions for guiding the co-evolution process.

4.1 Synchronization

In the beginning the target dataset is derived (as a slice or excerpt) from the source dataset, thus the following requirement always holds.
Requirement 1 (Initial Inclusion) At the initial time $t_0$, the target dataset $T$ is a subset of the source dataset $S$: $T_{t_0} \subseteq S_{t_0}$, and thus source and target datasets are in sync.

After some time, both source and target datasets evolve. At time $t_i$, the target dataset is $T_{t_i} = T_{t_0} \cup \delta(T_{t_0} - t_i)$ and the source dataset is $S_{t_i} = S_{t_0} \cup \delta(S_{t_0} - t_i)$.

Requirement 2 (Required Synchronization) At time $t_j$, a synchronization of both datasets is required iff source and target datasets were synchronised at time $t_i$, and the changesets applied to source and target datasets differ, i.e. $\delta(S_{t_i} - t_j) \neq \delta(T_{t_i} - t_j)$.

4.2 Conflict

When we synchronize the target $T_{t_i}$ with source $S_{t_i}$, there may exist triples which have been changed in both datasets. These changed triples may be conflicting.

Definition 5 (Potential Conflict). Let us assume that a synchronization is required for a given time slot $t_i - t_j$. $\delta(S_{t_i} - t_j)$ is the changeset of the source dataset and $\delta(T_{t_i} - t_j)$ is the changeset of the target dataset. A potential conflict is observed when there are triples $x_1 = (s, p, o_1) \in S_{t_j} \land x_2 = (s, p, o_2) \in \delta(T_{t_i} - t_j) \land x_2 \notin S_{t_j} = S_{t_i} \cup \delta(S_{t_i} - t_j)$ with $o_1 \neq o_2$.

Taking $o_1 \neq o_2$ as an indication for a conflict is subjective; in the sense that the characteristics of the involved property $p$ influences the decision. Consider two triples $(s, p, o_1)$ and $(s, p, o_2)$. If $p$ is a functional data type property, two triples are conflicting iff the object values $o_1$ and $o_2$ are not equal. However, if the property $p$ is a functional object property, these two triples are conflicting if the objects are or can be inferred to be different (e.g. via owl:differsFrom). Another property which needs special consideration is rdf:type. For this property it is necessary to check whether $o_1$ and $o_2$ belong to disjoint classes. Only then these triples would be conflicting. For example, si rdf:type Person and si rdf:type Athlete are not conflicting if Athlete is a subclass of Person (i.e. not disjoint). Thus, the process of detecting conflicts is considering the inherent characteristics of the involved property.

4.3 Synchronization Strategies

In the following, we list possible strategies for synchronization. We consider the time frame $t_i - t_j$, where in the time $t_i$, the source and target datasets are synchronised and until time $t_j$, both source and target datasets have been evolving independently. Before applying synchronization, the state of the source dataset is $S_{t_j} = S_{t_i} \cup \delta(S_{t_i} - t_j)$ and the target dataset is $T_{t_j} = T_{t_i} \cup \delta(T_{t_i} - t_j)$. 
**Strategy I:** This synchronization strategy prefers the source dataset and ignores all local changes on the target dataset; thus, the following requirement is necessary.

**Requirement 3 (Inclusion for synchronization)** *At any given time* $t_j$, after synchronising using selected strategy, the target dataset should be a subset of the source dataset, i.e. $T_{t_j} \subseteq S_{t_j}$.

Therefore, the target dataset ignores all triples $\{x \mid x \notin \delta(S_{t_i-t_j}) \land x \in \delta(T_{t_i-t_j})\}$ and adds only the triples $\{y \mid y \in \delta(S_{t_i-t_j})\}$. After synchronization, the state of the source dataset is $S_{t_j} = S_{t_i} \cup \delta(S_{t_i-t_j})$ and the state of the target dataset is $T_{t_j} = T_{t_i} \cup \delta(S_{t_i-t_j})$. Thus, the requirement 3 is met and $T_{t_j} \subseteq S_{t_j}$.

A special case of this strategy is when the target is not evolving.

**Example 2.** Applying strategy I for synchronization on Example 1 gives the following triples:

```
dbr:Adrian_Sanders rdf:type dbo:Politician;  
rdf:type dbo:Agent;  
foaf:name Adrian Sanders;  
dbp:spouse Alison Sanders;  
dbp:birthYear 1959-01-01 (xsd:date);  
dbp:birthYear 1959;  
owl:sameAs Freebase:Adrian Sanders;  
owl:sameAs http://wikidata.org/entity/Q479047.
```

**Strategy II:** With this strategy, the target dataset is not synchronized with the source dataset and keeps all its local changes. Thus, the target dataset is not influenced by any change from the source dataset and evolves locally. After synchronization, at time $t_j$, the state of the target dataset is $T_{t_j} = T_{t_i} \cup \delta(T_{t_i-t_j})$, and the state of the source dataset is $S_{t_j} = S_{t_i} \cup \delta(S_{t_i-t_j})$. It allows for synchronized replicas only if data is deleted. There is no synchronization if triples in the target dataset are updated or new triples are included.

**Example 3.** Applying strategy II for synchronization on Example 1 gives the following triples:

```
dbr:Adrian_Sanders rdf:type dbo:Politician;  
rdf:type dbo:MemberOfParliament;  
foaf:name Adrian Sanders;  
foaf:name Sanders, Adrian;  
dbp:birthYear 1959-01-01 (xsd:date);  
dbp:birthYear 1959;  
owl:sameAs http://yago-knowledge.org/resource/Adrian _Sanders.
```

**Strategy III:** This synchronization strategy respects the changesets of both source and target datasets except that it ignores conflicting triples.

Here, the set of triples in which conflicts occur is $X = \{(x_1 = (s, p, o_1) \in S_{t_j} \land x_2 = (s, p, o_2) \in \delta(T_{t_i-t_j}) \land x_2 \notin S_{t_j} \mid o_1 \neq o_2)\}^\mathbf{10}$ With Strategy

\[^\mathbf{10}\text{Set of conflicting triples selected after considering the inherent characteristics of the involved property. In rest of the paper, we say potential conflict a conflict, unless otherwise specified.} \]
III, the set of conflicting triples $X$ is removed from the target dataset while the source changeset $\delta(S_{t_i \rightarrow t_j})$ and the target changeset $\delta(T_{t_i \rightarrow t_j})$ are added. After synchronization, the state of the source dataset is $S_{t_j} = (S_{t_i} \cup \delta(S_{t_i \rightarrow t_j})) \setminus X$ and the state of the target dataset is $T_{t_j} = (T_{t_i} \cup \delta(T_{t_i \rightarrow t_j})) \setminus X$. Thus, requirement 3 is met.

**Example 4.** Applying strategy III for synchronization on Example 1 gives the following triples:

| dbr:Adrian_Sanders | rdf:type | dbo:Politician; |
|---------------------|----------|-----------------|
|                     | rdf:type | dbo:Agent;      |
|                     | rdf:type | dbo:MemberOfParliment; |
|                     | owl:sameAs | http://wikidata.org/entity/Q479047; |
|                     | owl:sameAs | http://yago-knowledge.org/resource/Adrian_Sanders. |

**Strategy IV:** This synchronization strategy also respects the changesets of both source and target datasets. In addition, it includes conflicting triples after resolving the conflicts.

Here, we consider the set of triples in which conflict occurs as $X = \{x_1 = (s, p, o_1) \in S_{t_j} \land x_2 = (s, p, o_2) \in \delta(T_{t_i \rightarrow t_j}) \land x_2 \notin S_{t_j} \text{ with } o_1 \neq o_2\}$. The conflicts over these triples should be resolved. It can be resolved using some resolution policy as described in [3]. Table 1 shows a list of various policies for resolving the conflicts. Conflict resolution results in a new set of triples called $Y$ whose triples are originated from $X$ but their conflicts have been resolved. Then, this new set (i.e. $Y$) is added to the both source and target datasets. After synchronization, the state of the source dataset is $S_{t_j} = ((S_{t_i} \cup \delta(S_{t_i \rightarrow t_j}) \cup \delta(T_{t_i \rightarrow t_j})) \setminus X) \cup Y$ and the state of target dataset is $T_{t_j} = ((T_{t_i} \cup \delta(T_{t_i \rightarrow t_j}) \cup \delta(S_{t_i \rightarrow t_j})) \setminus X) \cup Y$. Thus, requirement 3 is met.

**Example 5.** Applying strategy IV for synchronization on Example 1 while resolving the conflicts using function 'Any' gives the following triples:

| dbr:Adrian_Sanders | rdf:type | dbo:Politician; |
|---------------------|----------|-----------------|
|                     | rdf:type | dbo:Agent;      |
|                     | rdf:type | dbo:MemberOfParliment; |
|                     | foaf:name | Adrian Sanders; |
|                     | foaf:name | Sanders, Adrian; |
|                     | dbp:birthYear | 1959-01-01 (xsd:date); |
|                     | owl:sameAs | http://wikidata.org/entity/Q479047; |
|                     | owl:sameAs | http://yago-knowledge.org/resource/Adrian_Sanders. |

5 Approach

Our approach allows a user to choose a synchronization strategy (as presented in Section 4.3). Below, we describe the status of the source and target datasets after applying each synchronization strategy (see Algorithm 1).

Function $CDR$ is presented in Algorithm 2 which (i) identifies conflicts for the case of strategy III and strategy IV, and then (ii) resolves conflicts only in case of strategy IV. Our approach considers triple-based operations, explained below
using seven cases, to identify conflicts. Consider three triples \( x_1 = (s, p, o_1) \), \( x_2 = (s, p, o_2) \), and \( x_3 = (s, p, o_3) \) which are in conflict with each other \( x_1 \in \delta(S_{t_i,t_j}) \land x_2 \in \delta(T_{i_{-}t_j}) \land x_3 \in \{\delta(S_{t_i,t_j}) \land \delta(T_{i_{-}t_j})\} \land o_1 \neq o_2 \neq o_3 \). In the following we present seven cases of evolution causing conflicts. For the first three cases (I-III), the conflict resolution is straightforward. But for the cases IV-VII, we have to employ a conflict resolution policy to decide about triples \( x_1 \) and \( x_2 \):

- **Case I**: \( x_1 \) is added to \( T_{i_j} \) if \( x_1 \) is added by the source dataset and \( x_2 \) is deleted from the target dataset: \( x_1 \in \delta(S_{t_i,t_j})^+ \land x_2 \in \delta(T_{i_{-}t_j})^- \).
- **Case II**: \( x_1 \) is added to \( T_{i_j} \) if \( x_1 \) is modified by the source dataset and \( x_2 \) is deleted from the target dataset: \( x_1 \in \delta(S_{t_i,t_j})^+ \land x_2 \in \delta(S_{t_i,t_j})^- \land x_2 \in \delta(T_{i_{-}t_j})^- \).
- **Case III**: \( x_2 \) is added to \( S_{t_i} \) if \( x_1 \) is deleted from the source dataset and \( x_2 \) is modified in the target dataset: \( x_1 \in \delta(S_{t_i,t_j})^- \land x_2 \in \delta(T_{i_{-}t_j})^+ \land x_1 \in \delta(T_{i_{-}t_j})^- \).
- **Case IV**: if the triple \( x_1 \) is added to the source dataset and \( x_2 \) is added to the target dataset: \( x_1 \in \delta(S_{t_i,t_j})^+ \land x_2 \in \delta(T_{i_{-}t_j})^+ \).
- **Case V**: if \( x_3 \) is modified by both source and target datasets: \( x_2 \in \delta(S_{t_i,t_j})^+ \land x_3 \in \delta(S_{t_i,t_j})^- \land x_1 \in \delta(T_{i_{-}t_j})^+ \land x_3 \in \delta(T_{i_{-}t_j})^- \).
- **Case VI**: if \( x_1 \) is modified by the target dataset: \( x_1 \in \delta(S_{t_i,t_j})^+ \land x_2 \in \delta(T_{i_{-}t_j})^+ \land x_1 \in \delta(T_{i_{-}t_j})^- \).
- **Case VII**: if \( x_1 \) is modified by the source dataset: \( x_2 \in \delta(S_{t_i,t_j})^+ \land x_1 \in \delta(S_{t_i,t_j})^- \land x_2 \in \delta(T_{i_{-}t_j})^+ \).

### Table 1: Conflict resolution policies and functions

| Category          | Policy       | Function | Type | Description                                      |
|-------------------|--------------|----------|------|-------------------------------------------------|
|                   | Roll the dice| Any      | A    | Pick random value.                             |
|                   | Reputiation  | Best source | A   | Select the value from the preferred dataset.   |
|                   | Cry with the wolves | Global vote | A | Select the frequently occurring value for the respective attribute among all entities. |
|                   | Keep up-to-date | First*   | A   | Select the first value in order.              |
|                   |              | Latest*  | A   | Select the most recent value.                 |
|                   | Filter       | Threshold* | A   | Select the value with a quality score higher than a given threshold. |
|                   |              | Best*    | A   | Select the value with highest quality score. |
|                   |              | TopN*    | A   | Select the N best values.                     |
| Deciding          | Pass it on   | Concatenation | A | Concatenate all the values to get the resultant. |
|                   | Meet in the middle | Standard deviation, variance | N | Apply the corresponding function to get value. |
|                   |              | Average, median | N | Apply the corresponding function to get value. |
|                   |              | Sum      | N   | Select the sum of all values as the resultant. |
|                   | Trust your friends | Longest | S, C, T | Select the longest (non-NULL) value. |
|                   |              | Max      | N   | Select the maximum value from all.            |
|                   |              | Min      | N   | Select the minimum value from all.            |
|                   |              | Choose depending* | A | Select the value that belongs to a triple having a specific given value for another given attribute. |
|                   |              | Choose corresponding | A | Select the value that belongs to a triple whose value is already chosen for another given attribute. |
|                   |              | Most complete* | A | Select the value from the dataset (source or target) that has fewest NULLs across all entities for the respective attribute. |

- * requires metadata, A - All, S - String, C - Category (i.e., domain values have no order), T - Taxonomy (i.e., domain values have semi-order), N - Numeric.
Data: $S_i, T_i, \delta(T_i-t_j), \delta(S_i-t_j)$, strategy
Result: $S_{t_j}, T_{t_j}$

1. switch strategy do
  2. /* Synchronise with the source and ignore local changes */
  3. case Strategy I
  4. $T_{t_j} := T_{t_i} \cup \delta(S_{t_i-t_j})$;
  5. $S_{t_j} := S_{t_i}$;
  6. end

7. /* Do not synchronise with the source and keep local changes */
8. case Strategy II
9. $T_{t_j} := T_{t_i} \cup \delta(T_{t_i-t_j})$;
10. $S_{t_j} := S_{t_i} \cup \delta(S_{t_i-t_j})$;
11. end

12. /* Synchronise with the source and target datasets and ignore conflicts */
13. case Strategy III
14. $S_{t_j}, T_{t_j} := CDR(\delta(S_{t_i-t_j}), \delta(T_{t_i-t_j}), T_{t_i}, false)$;
15. end

16. /* Synchronise with the source and target datasets and resolve the conflicts */
17. case Strategy IV
18. $S_{t_j}, T_{t_j} := CDR(\delta(S_{t_i-t_j}), \delta(T_{t_i-t_j}), T_{t_i}, true)$;
19. end
20. endsw

Algorithm 1: Updating the source and target datasets by the chosen synchronization strategy.

Algorithm 2 shows the pseudocode of the procedure for updating the source and target datasets at the end of each timeframe. The function resolveConflict identifies operations described in Case I-VII. In addition, for the cases IV-VII, it resolves conflicts based on the type of involved predicate. As we discussed earlier, whether a conflict between two triple exists depends heavily on the type of property. Consider two triples $(s, p, o_1)$ and $(s, p, o_2)$, if $p$ is rdfs:label, we measure the similarity between $o_1$ and $o_2$ using the Levenshtein distance. We pick both values of rdfs:label if their similarity is below a certain threshold otherwise we treat them as conflicting. The function resolveConflict identifies operations containing deleted in the source, deleted/added/modified in the target dataset. In case of deleted in the source dataset and added/modified by the target dataset, it returns a triple to be added in $T_{t_j}$ otherwise null.

Figure 3 illustrates algorithm 2 for updating the target dataset $T_{t_j}$. We choose the synchronization strategy IV for the synchronization task. In the first step, we use a tree structure to identify conflicts for the triples in $\delta(S_{t_i-t_j})^*$. Consider the tree structure (a) in step 1 for the triple (dbr : Adrian_Sanders, rdf : type, dbo : Agent). We find different object values for (dbr : Adrian_Sanders, rdf : type) in $\delta(S_{t_i-t_j})^*$, $\delta(T_{t_i-t_j})^*$, and $T_{t_i}$. Then, we identify the triple based operation. For example, if we find the object value dbo : Agent in $\delta(S_{t_i-t_j})^*$, dbo : MemberOfParliment in $\delta(T_{t_i-t_j})^*$, and dbo : Politician in $T_{t_i}$, it represents case IV of addition by both source and target. Thus, this case represents a potential conflicting triple. We check if the values in $T_{t_i}$, $\delta(S_{t_i-t_j})^*$ and $\delta(T_{t_i-t_j})^*$ are disjoint for predicate rdf:type. As dbo : Politician, dbo : Agent, and dbo : MemberOfParliment are not disjoint, we pick all these values.
Data: $S_{t_j}, T_{t_j}, \delta(T_{t_i-t_j}), \delta(S_{t_i-t_j})$, conflict resolution

Result: $S_{t_j}, T_{t_j}$

1. $T_{t_j} = \phi$
2. $S_{t_j} = \phi$
3. temp = \phi
4. /* step1 */
5. for all triples $x_1 = (s_1, p_1, o_1) \in \delta(S_{t_i-t_j})^+$ do
6. /* finding triples which are in conflict with $x_1$ */
7. $X = \{x_2 = (s_1, p_1, Node.ANY) \in \delta(S_{t_i-t_j})^- \cup \delta(T_{t_i-t_j})^+ \cup \delta(T_{t_i-t_j})^- \cup T_{t_j}\}$
8. if $X = \phi$ then
9. temp = temp \cup x_1
10. end
11. else
12. $x = \text{resolveConflict}(x_1, X)$
13. temp = temp \cup x
14. end
15. end /* step1 */

16. /* step2 */
17. $T_{t_j} := T_{t_j} \setminus \delta(T_{t_i-t_j})^- \cup \delta(S_{t_i-t_j})^- ;$
18. $S_{t_j} := S_{t_j} \setminus \delta(T_{t_i-t_j})^- \cup \delta(S_{t_i-t_j})^- ;$
19. /* step3 */
20. temp := temp \cup \delta(S_{t_i-t_j})^+ \cup \delta(T_{t_i-t_j})^+$ ;
21. /* updating the target dataset */
22. $T_{t_j} := T_{t_j} \cup \text{temp}$
23. /* updating the source dataset */
24. $S_{t_j} := S_{t_j} \cup \text{temp}$

Algorithm 2: CDR algorithm: Conflict Detection and Resolution

Now, consider the tree structure (b) in step1 for triple (\textit{dbr} : Adrian\_Sanders, \textit{owl} : sameAs, \textit{http} : http://wikidata.org/entity/Q479047). It also represents case IV of addition by both source and target. The triple (\textit{dbr} : Adrian\_Sanders, \textit{owl} : sameAs, \textit{Freebase} : AdrianSanders) is added by source but deleted by target. Considering the target as more customized dataset, we give preference to target change. The tree structure (c) in step1 for the triple (\textit{dbr} : Adrian\_Sanders, \textit{dbp} : birthYear, 1959). It is also handled in case IV. As dbp:birthYear is functional property, we select only one value among already existing value and the new value using resolution function ‘Any’.

Furthermore, the user has the opportunity to adopt the manual or automatic selection of resolution functions. The resolution function is oriented to the type of predicates. The list of supported resolution functions is shown in Table 1. For automatic selection of conflict resolution functions for predicates, we check attributes of predicates (e.g., type, cardinality). Based on the usage analysis of different functions in [4], we prefer functions such as first, longest, and maximum for resolving conflicts. For instance, we prefer function longest for strings to avoid loss of information. For numeric data types, we prefer function max to keep the up-to-date value. For URIs, we pick the first value.
6 Evaluation

In order to assess the discussed approaches for synchronization and conflict identification/resolution, we prepare a testbed based on a slice of DBpedia using the following SPARQL query.

```
CONSTRUCT WHERE {
  ?s a Politician ;
  foaf:name ?name ;
  dbo:nationality ?nationality ;
  dbo:abstract ?abstract ;
  dbp:party ?party ;
  dbp:office ?office ;
  OPTIONAL { ?s foaf:depiction ?depiction } }
```

The extracted dataset is used as the initial source and target dataset. Then, we collect a series of changesets from DBpedia-live published from September 01, 2015 to October 31, 2015 using iRap [8]. We found a total of 304 changesets. These changesets are leveraged to simulate updates of the source and target datasets. We randomly select a total of 91 addition parts of changesets and altered values of their triples. Table 2 provides the number of triples of initial target, source and their associated changesets before synchronization. Initially, we have 200082 triples with 163114 unique objects in $T_t$ where $t_i = \text{September}01, 2015$.

![Fig. 3: Execution of algorithm 2 to synchronize $T_t$ with $S_{t_i}$](image)
Table 2: Number of triples in the source, target, and changesets for a given timeframe

| Scenario | \( |S_{ti} - T_{ti}| \) | \( |S_{ti} - T_{ti}| \) | \( |S_{ti} - T_{ti}| \) | \( |S_{ti} - T_{ti}| \) | Conflicting triples | RunTime (seconds) |
|----------|----------------|----------------|----------------|----------------|----------------|-----------------|
| 1        | 0              | 0              | 948            | 160            | -              | 0.0             |
| 2        | 0              | 0              | 11725          | 81             | -              | 0.0             |
| 3        | 11682          | 81             | 12060          | 81             | 343            | 0.5             |
| 4        | 11800          | 195            | 12186          | 81             | 343            | 2.0             |
| 5        | 5227           | 131            | 6081           | 121            | 186            | 0.2             |

The running time of the five different scenarios is also shown in Table 3 (These times are recorded only for the execution of synchronization part and do not include data loading time). Evaluation showed that strategy IV (performed in scenario IV) needs more time even from strategy III (performed in scenario III) where all conflicts were detected but not resolved.

Synchronization influences data quality specially in terms of data consistency. To evaluate the usefulness of the synchronization approach, we use three data quality metrics i.e. (1) completeness, (2) conciseness, and (3) consistency described as follows:

1. Completeness refers to the degree to which all required information is present in a dataset [24]. We measure it for source and target changesets to identify which helps more in completeness. We measure it using

\[
\frac{\text{Number of unique triples in synchronised dataset}}{\text{Number of unique triples in (initial dataset} \cup \text{ changeset)}}
\]

2. Consistency states that the values should not be conflicting. We measure it using

\[
\frac{\text{Number of non-conflicting triples in synchronized dataset}}{\text{Number of triples in (initial dataset} \cup \text{ source and target changesets)}}
\]

3. Conciseness measures the degree to which the dataset does not contain redundant information using

\[
\frac{\text{Number of unique triples in dataset}}{\text{Number of all triples in dataset}}
\]
Conciseness (before synchronization) is computed using initial target dataset and source and target changesets. We compute these metrics for all the assumed scenarios, the results are shown in Table 4. For our sample case study, we found almost equal contribution of both source and target changesets in reducing the missing information. However, we found minimum 163191 number of unique objects using strategy II and maximum 163591 number of unique objects using strategy IV. Please note that strategy 1 and strategy II may not necessarily increase the number of unique triples as they do not consider about conflicts. It can be observed by analyzing the scenario 1 where the role of source changesets in completeness is 99% which is less than the target contribution. Through evaluation, we found significant increase in conciseness for all strategies.

Table 4: Synchronization effect on completeness, consistency, and conciseness

| Scenario | Completeness (source) | Completeness (target) | Consistency | Conciseness (before synchronization) | Conciseness (after synchronization) |
|----------|------------------------|------------------------|-------------|--------------------------------------|-------------------------------------|
| 1        | 99%                    | 100%                   | -           | 77%                                  | 81%                                 |
| 2        | 99%                    | 99%                    | -           | 77%                                  | 81%                                 |
| 3        | 99%                    | 100%                   | 94%         | 77%                                  | 81%                                 |
| 4        | 99%                    | 100%                   | 94%         | 77%                                  | 81%                                 |
| 5        | 99%                    | 100%                   | -           | 77%                                  | 81%                                 |

7 Related Work

Related work includes synchronization of semantic stores for concurrent updates by autonomous clients [1], synchronization of source and target [22], replication of partial RDF graphs [19], ontology change management [12], and conflict resolution for data integration [3–5,11,13,14,16,17,21]. We discuss related work here along the dimensions change management and conflict resolution.

7.1 Change Management

Efficient synchronization of semantic stores is challenging due to the factors, scalability and number of autonomous participants using replica. C-Set [1] is a Commutative Replicated Data Type (CRDT) that allows concurrent operations to be commutative and thus, avoids other integration algorithms for consistency. The approach, proposed in [19], allows to replicate part of an RDF graph on clients. Clients can apply offline changes to this partial replica and write-back to original data source upon reconnection. Table 5 provides a comparative analysis of change management approaches used for synchronization.

A few surveyed approaches [2,12] are related to ontological change management. In [12], a framework is developed for ontology change management and tested for RDF ontologies. This framework allows to design ontology evolution algorithms. In [2], an approach for the versioning and evolution of ontologies, based on RDF data model, is presented. It considers atomic changes such as addition or deletion of statement and then aggregates them to compound changes to form a change hierarchy. This change hierarchy allows human reviewers to analyze at various levels of details.
7.2 Conflict resolution

For relational databases, there is much work on inconsistency resolution [3, 4, 16]. The Humboldt Merger [3], an extension to SQL with a FUSE BY statement, resolves conflicts at runtime. Fusionplex [16] integrates data from heterogeneous data sources and resolves inconsistencies during data fusion. For fusion, it uses parameters such as user-defined data utility, threshold of acceptance, fusion functions, and metadata. [4] classifies conflict resolution strategies into three classes: ignorance, avoidance, and resolution. Conflict ignorance strategies are not aware of conflicts in the data. Conflict avoidance strategies are aware of whether and how to handle inconsistent data. Conflict resolution strategies may use metadata to resolve conflicts. These can be divided into deciding and mediating. A deciding strategy chooses value from already existing values whereas a mediating strategy may compute a new value.

Sieve Fusion Policy Learner [5] uses a gold standard dataset to learn optimal fusion function for each property. The user specifies possible conflict resolution strategies from which the learning algorithm selects the one that gives maximum results within error threshold with respect to the gold standard.

Most relevant approaches to our proposed work are Sieve [13] - part of Linked Data integration framework (LDIF) [21], data fusion algorithm [14] for OD-CleanStore [11], RDFSync [22], and Col-graph [10]. Our approach differs from the previous ones in the scope of the problem (see Figure 4). RDFSync performs synchronization of two datasets by merging both graphs, deleting information which is not known by source, or making the target equal to source. In contrast to RDFSync, our co-evolution approach allows merging of both graphs while ignoring or resolving conflicts and keeping only source or target changes. Col-graph deals with consistent synchronization of replicas and does not tackle conflicts.

Sieve and ODCS are data fusion approaches and thus, are applicable where described data have different schemata. In contrast to both, co-evolution approach is applicable where described data have same schemata. Both approaches define conflicts as RDF triples sharing same subject/predicate with inconsistent values for objects. Sieve uses quality scores to resolve data while, ODCS produces quality scores of resolved data and keeps name of dataset from where the resolved value belongs. We extend the conflict definition by further considering the predicate type, as discussed earlier (see Definition 5).

| Approach   | Synchronization | Bi-directional | Participants | Conflict handling* |
|------------|-----------------|----------------|--------------|--------------------|
| C-Set      | ✓               | ✓              | ✓            | ✓                  |
| RDFSync    | ✓               | ✓              | source, target | x                  |
| Col-graph  | ✓               | ✓              | ✓            | ✓                  |
| [14]       | ✓               | ✓              | back to source | ✓                  |
| Co-evolution | ✓              | ✓              | source, target | ✓                  |

* - Triple level conflicts according to Definition 5
8 Conclusion and Future Work

In this paper we presented an approach to deal with co-evolution which refers to mutual propagation of the changes between a replica and its origin dataset. Using the co-evolution process, we address synchronization and conflict resolution issues. We demonstrated the approach using formal definitions of all the concepts required for realizing co-evolution of RDF datasets and implemented it using different strategies. We evaluated the approach using data quality metrics completeness, conciseness, and consistency. A thorough evaluation of the approach, using DBpedia changesets, indicates that our method can significantly improve the quality of dataset. In the future, we will extend the concept of conflict resolution at schema level. For example, renaming a class invalidates all triples that belong to it in a dataset. Further, we will evaluate the scalability and performance of our proposed approach using a benchmark dataset.

Acknowledgements. This work is supported in part by the European Union’s Horizon 2020 programme for the projects BigDataEurope (GA 644564) and WDAqua (GA 642795). Sidra Faisal is supported by a scholarship of German Academic Exchange Service (DAAD).

References

1. Aslan, K., Molli, P., Skaf-Molli, H., Weiss, S.: C-set: A commutative replicated data type for semantic stores. In: 4th Int. Workshop on REsource Discovery (RED) (2011)
2. Auer, S., Herre, H.: A versioning and evolution framework for rdf knowledge bases. In: 6th Int. Conf. on Perspectives of Systems Informatics (PSI). pp. 55–69 (2006)
3. Bilke, A., Bleiholder, J., Naumann, F., Böhm, C., Draba, K., Weis, M.: Automatic data fusion with hummer. In: 31st Int. Conf. on Very Large Data Bases (VLDB) (2005)
4. Bleiholder, J., Naumann, F.: Data fusion and conflict resolution in integrated information systems. In: Int. Workshop on Information Integration on the Web (2006)
5. Bryl, V., Bizer, C.: Learning conflict resolution strategies for cross-language wikipedia data fusion. In: 23rd Int. Conf. on World Wide Web (WWW) (2014)
6. Buil-Aranda, C., Hogan, A., Umbrich, J., Vandenbussche, P.: SPARQL web-querying infrastructure: Ready for action? In: The Semantic Web - ISWC 2013
7. Endris, K.M., Faisal, S., Orlandi, F., Auer, S., Scerri, S.: Interest-based RDF update propagation. In: The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part I. pp. 513–529 (2015)

8. Endris, K.M., Faisal, S., Orlandi, F., Auer, S., Scerri, S.: irap - an interest-based rdf update propagation framework. In: ISWC 2015 Posters and Demonstrations Track co-located with the 14th Int. Semantic Web Conf. (ISWC) (2015)

9. Feigenbaum, L., Williams, G., Clark, K., Torres, E.: SPARQL 1.1 protocol (2013), http://www.w3.org/TR/sparql11-protocol/

10. Ibanez, L.D., Skaf-Molli, H., Molli, P., Corby, O.: Col-graph: Towards writable and scalable linked open data. In: 13th Int. Semantic Web Conf. (ISWC) (2014)

11. Knap, T., Michelfeit, J., Daniel, J., Jerman, P., Rychnovský, D., Soukup, T., Nečaský, M.: Odcleanstore: A framework for managing and providing integrated linked data on the web. In: Web Information Systems Engineering (WISE) (2012)

12. Konstantinidis, G., Flouris, G., Antoniou, G., Christophides, V.: Ontology evolution: A framework and its application to RDF. In: Joint ODBIS-SWDB Workshop on Semantic Web, Ontologies, Databases (2007)

13. Mendes, P.N., Mülleisen, H., Bizer, C.: Sieve: Linked data quality assessment and fusion. In: Joint EDBT-ICDT Workshops, pp. 116–123 (2012)

14. Michelfeit, J., Knap, T., Nečaský, M.: Linked data integration with conflicts. Web Semantics (2014)

15. Montoya, G., Skaf-Molli, H., Molli, P., Vidal, M.E.: Federated sparql queries processing with replicated fragments. In: The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part I (2015)

16. Motro, A., Anokhin, P.: Fusionplex: Resolution of data inconsistencies in the integration of heterogeneous information sources. Information fusion 7(2) (2006)

17. Paton, N.W., Christodoulou, K., Fernandes, A.A.A., Parsia, B., Hedele, C.: Pay-as-you-go data integration for linked data: Opportunities, challenges and architectures. In: 4th Int. Workshop on Semantic Web Information Management (2012)

18. Saleem, M., Ngomo, A.C.N., Parreíra, J.X., Deus, H.F., Hauswirth, M.: Daw: Duplicate-aware federated query processing over the web of data. In: 12th Int. Semantic Web Conf. (ISWC) (2013)

19. Schandl, B.: Replication and versioning of partial RDF graphs. In: 7th Int. Conf. on The Semantic Web, pp. 31–45 (2010)

20. Schmachtenberg, M., Bizer, C., Paulheim, H.: Adoption of the linked data best practices in different topical domains. In: The Semantic Web - ISWC 2014. pp. 245–260 (2014)

21. Schultz, A., Matteini, A., Isele, R., Bizer, C., Becker, C.: Ldif – linked data integration framework. In: 2nd Int. Workshop on Consuming Linked Data (2011)

22. Tummarello, G., Morbidoni, C., Bachmann-Gmür, R., Erling, O.: Rdfsync: Efficient remote synchronization of RDF models. In: 6th Int. The Semantic Web and Second Asian Conf. on Asian Semantic Web Conference (ISWC-ASWC) (2007)

23. Verborgh, R., Hartig, O., Meester, B.D., Haesendonck, G., Vocht, L.D., Sande, M.V., Cyganiak, R., Colpaert, P., Mannens, E., de Walle, R.V.: Querying datasets on the Web with high availability. In: ISWC. pp. 180–196 (2014)

24. Zaveri, A., Rula, A., Maurino, A., Pietrobon, R., Lehmann, J., Auer, S.: Quality assessment for linked open data: A survey. Semantic Web Journal (2015)