On the Foundations of Computing Deltas Between RDF Models

Dimitris Zeginis, Yannis Tzitzikas, and Vassilis Christophides

Computer Science Department, University of Crete, GREECE, and
Institute of Computer Science, FORTH-ICS, Greece
{zeginis,tzitzik,christop}@ics.forth.gr

Abstract. The ability to compute the differences that exist between two RDF models is an important step to cope with the evolving nature of the Semantic Web (SW). In particular, RDF Deltas can be employed to reduce the amount of data that need to be exchanged and managed over the network and hence build advanced SW synchronization and versioning services. By considering Deltas as sets of change operations, in this paper we study various RDF comparison functions in conjunction with the semantics of the underlying change operations and formally analyze their possible combinations in terms of correctness, minimality, semantic identity and redundancy properties.

1 Introduction

In order to cope with the evolving nature of the Semantic Web (SW) we need effective and efficient support for building advanced SW synchronization and versioning services. RDF Deltas, reporting the differences that exist between two RDF models have been proven to be crucial in order to reduce the amount of data that need to be exchanged and managed over the network in this respect [17,18,3,8].

Although RDF models can be serialized in various text formats (e.g., XML[1], N-Triples[2], Trix[3]), a straightforward application of existing version control systems for software code, such as RCS [29] and CVS [4], is not a viable solution for computing RDF Deltas. This is mainly due to the fact that RDF models, essentially represent graphs which (a) may feature several possible serializations (since there is no notion of edge ordering in [4]) and (b) are enriched with the semantics of RDFS specification (also including inferred edges according to [5]). For these reasons, several non text-based tools have been recently developed for comparing RDF graphs produced autonomously on the SW, as for example, SemVersion [31], PromptDiff [23], Ontoview [18], [10] and [3]. In most cases, the output of these tools is exploited by humans, and thus an intuitive presentation of the comparison results (and other related issues) has received considerable attention. SemVersion [31] proposes two Diff algorithms: (a) one *structure-based* which returns a set-based difference of the triples explicitly forming the two graphs, and (b) one *semantic-aware* which also takes into account the triples inferred by the associated RDFS schemas. PromptDiff [23,24,22] is an ontology-versioning environment,

[1] http://www.w3.org/TR/rdf-syntax-grammar/
[2] http://www.w3.org/2001/sw/RDFCore/ntriples/
[3] http://www.w3.org/2004/03/trix/

K. Aberer et al. (Eds.): ISWC/ASWC 2007, LNCS 4825, pp. 637–651, 2007.
© Springer-Verlag Berlin Heidelberg 2007

that includes a version-comparison algorithm (based on heuristic matchers [23,24]), while the visualization of the computed difference between two ontologies is discussed in [22]. Ontoview [18] is an ontology management system, able to compare two ontology versions and highlight their differences. Notably, it allows users to specify the conceptual relations (i.e. equivalence, subsumption) between the different versions of an ontology concept. Moreover, [10,13] introduce the notion of RDF molecules as the finest components to be used when comparing RDF graphs (in the absence of blank nodes each triple constitutes a molecule). Finally, tracking the evolution of ontologies when changes are preformed in more controlled environments (e.g. collaborative authoring tools) has been addressed in [19,25,32].

However, existing RDF comparison tools have not yet focused on the size of the produced Deltas, a very important aspect for building versioning services over SW repositories [28]. In this paper we are interested in computing RDF Deltas as sets of change operations (i.e. SW update programs) that enable us to transform one RDF model into the other. Consider, for example, the two RDF models K and K' of Figure 1 and their standard representation as sets of explicitly defined triples [5]: what set of change operations could transform K to K' ($\Delta(K \rightarrow K')$) or vice versa ($\Delta(K' \rightarrow K)$)?

To answer this question we need to consider the semantics of the update primitives such as $Add(t)$ and $Del(t)$ where t is triple involving any RDF predicate. By assuming a side-effect free semantics for these primitives, i.e. $Add(t)$ (resp. $Del(t)$) is a straightforward addition (resp. deletion) of t from the set $Triples(K)$, K' can be obtained by executing the following set Δ_e (e stands for $e$xplicit) of change operations:

$$\begin{aligned}\Delta_e = \{&Del(TA \textit{ subClassOf } Person),\ Del(Address \textit{ domain } Student),\\ &Del(Jim \textit{ type } Student),\ Add(TA \textit{ subClassOf } Student),\\ &Add(Address \textit{ domain } Person),\ Add(Jim \textit{ type } Person)\}\end{aligned}$$

Δ_e is actually composed of update operations over the explicit triples of K and K', and it is provided by the majority of existing RDF comparison tools [3,31,10]. However, by assuming side-effects (on the inferred triples not represented in Figure 1) during the execution of the above update primitives, we can reach K' by applying on K the following set Δ_d (d stands for $d$ense) of change operations:

$$\begin{aligned}\Delta_d = \{&Del(Jim \textit{ type } Student),\ Add(TA \textit{ subClassOf } Student),\\ &Add(Address \textit{ domain } Person)\}\end{aligned}$$

As we can easily observe, Δ_d has only three change operations in contrast to Δ_e that has six, given that inferred triples are also taken into account for the Delta computation. For example, $Del(TA \textit{ subClassOf } Person)$ is not included in Δ_d because it can be inferred from K'. As we can see in Figure 1, this comparison function yields even smaller in size operation sets than the Δ_c (c stands for $c$losure) semantics-aware Delta of [31]. However, Δ_d cannot always successfully transform one RDF model to another. Returning to our example of Figure 1, Δ_d cannot be used to migrate backwards from K' to K since $Del(Address \textit{ domain } TA)$ is an operation not included in Δ_d . For this reason, we need to consider additional RDF comparison functions involving inferred triples such as Δ_{dc} (dc stands for $d$ense & $c$losure) illustrated in Figure 1. Still the resulting sets of operations have at most the same size as those returned by Δ_c.

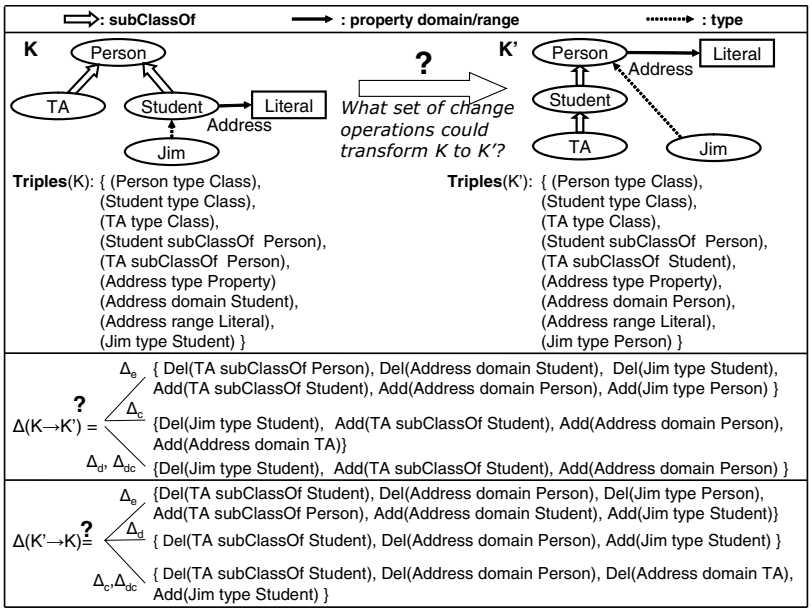


Fig. 1. Transforming K to K' and vice versa

RDF comparison functions that yield as less as possible change operations are quite beneficial for building SW versioning services. In particular, by advocating a change-based versioning framework [9] we can store in a SW repository only the update programs required to migrate (forward or backward) from one version to another rather than the entire set of triples for each version. In a nutshell, storing (or exchanging) as less as possible change operations is more space (or time) efficient. In this context, the main questions addressed by our work are: (a) what semantics of update primitives would make the above scenario possible (i.e. with what side-effects), and (b) how could we compute the corresponding set of change operations (i.e. with what comparison functions)? In response to these questions, the main contributions of this paper are:

(a) We introduce two change operations semantics: one *plain* set-theoretic (considers only updates of the explicit triples) denoted by $\mathcal{U}_p$, and the other involves *inference* and *redundancy elimination* of updated Knowledge Bases, denoted by $\mathcal{U}_{ir}$.
(b) We analyze four different comparison functions returning sets of changes operations, namely, *explicit* (Δ_e), *closure* (Δ_c), *dense* (Δ_d), and *dense & closure* (Δ_{dc}).
(c) We study which combinations of change operation semantics and comparison functions are correct and satisfy properties such as semantic identity and non redundancy. It should be stressed that the combination $(\Delta_{dc}, \mathcal{U}_{ir})$ is quite promising: (i) it returns an empty result if K and K' are semantically equivalent (ii) the knowledge base obtained when applying $\Delta_{dc}(K \rightarrow K')$ on K is redundancy free, and (iii) if K' is an extension of K then it is guaranteed that the Delta that we get is smaller than all comparison functions already proposed in the literature [3,31,10].

The rest of this paper is organized as follows. Section 2 provides background information regarding RDF Knowledge Bases (KB). Section 3 introduces four RDF comparison functions, Section 4 elaborates on the change operations and their semantics, while Section 5 shows the interplay between the two. Finally, Section 6 concludes the paper and identifies issues for further research.

2 Background: RDF KBs

In general, an RDF Knowledge Base (KB) is defined by a set of triples of the form (subject, predicate, object). Let $\mathcal{T}$ be the set of all possible triples that can be constructed from an infite set of URIs (for resources, classes and properties) as well as literals [15]. Then a KB can be seen as a finite subset K of $\mathcal{T}$, i.e. $K \subseteq \mathcal{T}$. Apart from the explicitly specified triples of a K, other triples can be inferred based on the semantics of RDF/S [16]. For this reason, we introduce the notion of closure and reduction of RDF KBs.

The *closure* of a K, denoted by $C(K)$, contains all the triples that either are explicitly specified or can be inferred from K by taking in account the semantics of the associated RDFS schemas. As we can view an RDF model as a graph, we could consider that $C(K)$ is defined (and computed) by taking the reflexive and transitive closures of binary relations (subsumption, type)[4]. If it holds $C(K) = K$, then we will call K *completed*. The elements of K will be called *explicit* triples, while the elements of $C(K) - K$ will be called *inferred*. We can now define an equivalence relation between two knowledge bases.

Def. 1. Two knowledge bases K and K' are *equivalent*, denoted by $K \sim K'$, iff $C(K) = C(K')$.

The *reduction* of a K, denoted by $R(K)$, is the smallest in size set of triples such that $C(R(K)) = C(K)$. In general, the reduction of a K is not necessarily unique (when cycles occur in the subsumption relations). Let Ψ denote the set of all knowledge bases that have a unique reduction. Independently of whether the reduction of a K is unique or not, we can characterize a K as (semantically) *redundancy free*, and we can write $RF(K) = True$ (or just $RF(K)$), if it does not contain explicit triples which can be inferred from K. Formally, K is redundancy free if there is not any proper subset K' of K (i.e. $K' \subset K$) such that $K \sim K'$.

3 RDF KBs Deltas

In this section we formally define the four comparison functions of RDF KBs introduced in Figure 1, namely, Δ_e, Δ_c, Δ_d and Δ_{dc}.

$$\Delta_e(K \to K') = \{Add(t) \mid t \in K' - K\} \cup \{Del(t) \mid t \in K - K'\}$$
$$\Delta_c(K \to K') = \{Add(t) \mid t \in C(K') - C(K)\} \cup \{Del(t) \mid t \in C(K) - C(K')\}$$
$$\Delta_d(K \to K') = \{Add(t) \mid t \in K' - C(K)\} \cup \{Del(t) \mid t \in K - C(K')\}$$
$$\Delta_{dc}(K \to K') = \{Add(t) \mid t \in K' - C(K)\} \cup \{Del(t) \mid t \in C(K) - C(K')\}$$

[4] The consequence operator of logic theories (e.g. see [11]) is out of the scope of this paper.

Δ_e (where e stands for $e$xplicit) actually returns the triple-set difference over the explicitly specified triples, while Δ_c (where c stands for $c$losure) returns the triple-set difference by also taking into account the inferred triples. As we mentioned in Section 1, existing approaches (e.g. [31]) are based on Δ_e and Δ_c. However, as we are especially interested in comparison functions that yield smaller in size Deltas, we introduce two novel comparison functions namely Δ_d (where d comes from $d$ense) and Δ_{dc} (dc comes from $d$ense & $c$losure). It is not hard to see that Δ_d yields smaller in size outputs (in comparison with the previous two). Unfortunately, and as we will see at Section 5, Δ_d cannot be used in general since only for specific cases returns correct results. For this reason we additionally consider Δ_{dc} which yields smaller in size outputs than Δ_c. This function resembles Δ_d regarding additions and Δ_c regarding deletions.

Prop. 1. Let $|\Delta(K \rightarrow K')|$ to denote the *number* of change operations in $\Delta(K \rightarrow K')$. Then for any pair of valid knowledge bases K and K' it holds:

$$|\Delta_d(K \rightarrow K')| \leq |\Delta_e(K \rightarrow K')|$$
$$|\Delta_d(K \rightarrow K')| \leq |\Delta_{dc}(K \rightarrow K')| \leq |\Delta_c(K \rightarrow K')|$$

We have $K \subseteq C(K) \Leftrightarrow K' - C(K) \subseteq K' - K$ (1) and $K' \subseteq C(K') \Leftrightarrow K - C(K') \subseteq K - K'$ (2). From (1) and (2) it follows that $|\Delta_d| \leq |\Delta_e|$. The formula for additions is the same for both Δ_d and Δ_{dc}. If we consider deletions we have $K \subseteq C(K) \Leftrightarrow K - C(K') \subseteq C(K) - C(K') \Leftrightarrow |\Delta_d| \leq |\Delta_{dc}|$. Furthermore, we have $K' \subseteq C(K') \Leftrightarrow K' - C(K) \subseteq C(K') - C(K)$ (3) and $K \subseteq C(K) \Leftrightarrow K - C(K') \subseteq C(K) - C(K')$ (4). From (3) and (4) it follows that $|\Delta_d| \leq |\Delta_c|$. Finally, the formula for deletions is the same for both Δ_c and Δ_{dc}. If we consider additions we have $K' \subseteq C(K') \Leftrightarrow K' - C(K) \subseteq C(K') - C(K) \Leftrightarrow |\Delta_{dc}| \leq |\Delta_c|$

In a nutshell Δ_d gives always smaller in size Deltas while Δ_{dc} is incomparable to Δ_e (Figure 2 shows the Hasse diagram of the ordering relation).

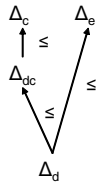


Fig. 2. Ordering of Comparison Functions with respect to the size of their output

In the next section we will investigate what happens if we "execute" the Deltas produced by the above comparison functions under different semantics of the change update primitives $Add(t)$ and $Del(t)$.

4 RDF KB Change Operations Semantics

A *change operation semantics* defines precisely the pre and post-conditions of the operations $Add(t)$, $Del(t)$ where t is a triple involving any RDF predicate. In Table 1

we define two alternative semantics, namely, $\mathcal{U}_p$ (p comes from $p$lain), and $\mathcal{U}_{ir}$ (ir comes from $i$nference & $r$eduction). Under $\mathcal{U}_p$-semantics, the execution of the operations consists of plain set theoretic additions and deletions of triples. This implies that only the explicit triples are taken into account while inferred ones are ignored. Under $\mathcal{U}_{ir}$-semantics the execution of update primitives incurs also interesting side-effects such as redundancy elimination and knowledge preservation. This implies that the updated KB will not contain any explicit triple which can be inferred, while preserves as much of the knowledge expressed in K as possible (reminiscent to the postulates of the AGM theory [2] regarding contraction, and compliant with the semantics of the RUL update language [21]).

We first explain $\mathcal{U}_{ir}$ using the example of Figure 1. If we apply on K the set Δ_d under $\mathcal{U}_{ir}$-semantics, then we will indeed get K'. The insertion of (TA ***subClassOf*** $Student$) makes the triple (TA ***subClassOf*** $Person$) redundant, so the execution of $Add(TA$ ***subClassOf*** $Student)$ will remove (TA ***subClassOf*** $Person$) from the KB. Analogously, the insertion of ($Address$ ***domain*** $Person$) makes the triple ($Address$ ***domain*** $Student$) redundant, while the deletion of the triple (Jim ***type*** $Student$) will add the triple (Jim ***type*** $Person$).

Returning to Table 1, for every operation u (of the form $Add(t)$ or $Del(t)$) three different, and mutually exclusive, pre-conditions are examined, namely $t \in K$, $t \in C(K) - K$ and $t \notin C(K)$. The post-conditions of each case are specified. K (K') denotes the knowledge base before (after) the execution of an operation u. Notice that post-conditions define exactly what K' will be[5], unless the reduction is not unique.

Table 1. Two change operation semantics $\mathcal{U}_p$ and $\mathcal{U}_{ir}$

Change Operation Semantics $\mathcal{U}_p$				
Operation		Pre-condition	Post-condition	Comment‘
$Add(t)$	1	$t \in K$	$K' = K$	void
	2	$t \in C(K) - K$	$K' = K \cup \{t\}$	addition (although already inferred)
	3	$t \notin C(K)$	$K' = K \cup \{t\}$	addition
$Del(t)$	4	$t \in K$	$K' = K - \{t\}$	deletion
	5	$t \in C(K) - K$	$K' = K$	an inferred triple cannot be deleted
	6	$t \notin C(K)$	$K' = K$	void
Change Operation Semantics $\mathcal{U}_{ir}$				
$Add(t)$	7	$t \in K$	$K' = K$	void
	8	$t \in C(K) - K$	$K' = K$	it is already inferred so it is ignored
	9	$t \notin C(K)$	$K' = R(K \cup \{t\})$	addition and then reduction
$Del(t)$	10	$t \in K$	$K' = R(C(K) - \{t\})$	deletion from closure and then reduction
	11	$t \in C(K) - K$	$K' = K$	an inferred triple cannot be deleted
	12	$t \notin C(K)$	$K' = K$	void

In particular, let t be the triple whose addition is requested. If $t \in K$, then under both $\mathcal{U}_p$ and $\mathcal{U}_{ir}$ semantics no change will be made i.e. $K' = K$ (recall that K is a *set* of triples). If $t \in C(K) - K$, then under $\mathcal{U}_p$-semantics, K' will indeed contain that triple however, under $\mathcal{U}_{ir}$-semantics we will have $K' = K$ because every triple that

[5] One could consider the rows of Table 1 as ECA rules where the Events correspond to column “Operation”, the Conditions correspond to column “Pre-Condition” and the Actions correspond to column “Post-condition”.

exists at $C(K) - K$ can be inferred (and $\mathcal{U}_{ir}$ aims at redundancy-free KBs). Finally, when requesting the addition of a triple $t \notin C(K)$ under $\mathcal{U}_p$, K' will contain that triple. Under $\mathcal{U}_{ir}$, K' will contain the triples that remain after adding t to K and eliminating the redundant triples (i.e. those that can be inferred).

Let us now consider that the deletion of a triple t is requested. If t belongs to K, then K' will not contain t under $\mathcal{U}_p$-semantics. Under $\mathcal{U}_{ir}$, K' will contain the triples that remain after deleting t from $C(K)$ and eliminating the redundant triples (note that $C(K)$ is used in order to preserve as much knowledge as possible). Now if $t \in C(K) - K$, then this request is ignored under both semantics. This means that in both semantics, *only explicit triples can be deleted*. This relieves us from having to decide which of the (possibly several) policies to adopt for reaching a K' whose closure does not contain t. Finally, if $t \notin C(K)$, then nothing happens as t is already out of K.

Let $\mathcal{S}$ be the set of all possible operations of the form $Add(t)$, $Del(t)$ where $t \in \mathcal{T}$. Let S be a finite subset of $\mathcal{S}$ (i.e. $S \subset \mathcal{S}$). If $\mathcal{U}$ is a symbol that denotes the semantics of a particular change operation (i.e. $\mathcal{U}_p, \mathcal{U}_{ir}$), then we will use $S^{\mathcal{U}}(K)$ to denote the result of applying S to K under $\mathcal{U}$ semantics. Notice that the result of applying an operation is unique under $\mathcal{U}_p$-semantics. This is true also for $\mathcal{U}_{ir}$ if we are in Ψ (KBs with unique reduction).

Now we introduce some notions regarding *sets* of change operations (based on [30]). Two sets of change operations S and S' are *universally equivalent* under $\mathcal{U}$, denoted by $S \equiv^{\mathcal{U}} S'$, iff $S^{\mathcal{U}}(K) \sim S'^{\mathcal{U}}(K)$ for every possible knowledge base K.

For computing change-based Deltas we need a less strong notion of equivalence (analogously to transaction equivalence [1]).

Def. 2. S and S' are *equivalent over a given* K under $\mathcal{U}$, denoted by $S \equiv_K^{\mathcal{U}} S'$, iff $S^{\mathcal{U}}(K) \sim S'^{\mathcal{U}}(K)$.

In order to elaborate on cases where the order of execution of the update operations affects the final result, we introduce the following notion of satisfaction.

Def. 3. We will say that K *satisfies*: (a) an operation $Add(t)$, iff $t \in C(K)$, (b) an operation $Del(t)$, iff $t \notin C(K)$ and (c) a set of change operations S (where $S \subseteq \mathcal{S}$) if K satisfies *every* element of S.

If the resulting KB does not satisfy S, then we will write $S(K) = \mathcal{E}$ where $\mathcal{E}$ is a special symbol indicating that an error occurred. In the sequel, and for reasons of brevity, whenever we write $S(K)$ we will also mean that $S(K) \neq \mathcal{E}$.

5 Comparison Functions and Change Operation Semantics

In this section we investigate which of the four comparison functions (introduced in Section 3) and under what semantics of update primitives (presented in Section 4) could be used for building a change-based versioning system. To this end, we have to define formally the notions of *correctness*, *semantic identity* and *redundancy*, and then elaborate on the execution of the update programms. Finally, we will identify these pairs that are correct and the properties that they satisfy.

5.1 Correctness, Semantic Identify and Non Redundancy of RDF Deltas

Let Δ_x be a comparison function, and $\mathcal{U}_y$ be a change operation semantics.

Def. 4. A pair $(\Delta_x, \mathcal{U}_y)$ is *correct* if for any pair of knowledge bases K and K', it holds $\Delta_x(K \rightarrow K')^{\mathcal{U}_y}(K) \sim K'$.

Obviously, a pair $(\Delta_x, \mathcal{U}_y)$ can be used for versioning services only if it is correct. Apart from correctness, a pair $(\Delta_x, \mathcal{U}_y)$ may also satisfy the following properties.

(P1) If $K \sim K'$ then $\Delta_x(K \rightarrow K') = \emptyset$ (*semantic identity*)
It is desirable to have a comparison function that reports an empty result if its operands are equivalent.

(P2) $RF(\Delta_x(K \rightarrow K')^{\mathcal{U}_y}(K))$ (*non redundancy*)
The resulting KB is always redundancy free (i.e. for any K and K').

(P2.1) If $RF(K')$ then $RF(\Delta_x(K \rightarrow K')^{\mathcal{U}_y}(K))$
If K' is RF then the resulting KB is also RF. Note that (P2.1) is weaker than (P2): if (P2) holds then (P2.1) holds too.

5.2 Executing (or Satisfying) RDF Deltas

Def. 4 presupposes that we have at our disposal an appropriate "execution mode" such that when we apply $\Delta_x(K \rightarrow K')$ in K, and according to the selected semantics, the resulting KB will *satisfy* every element of $\Delta_x(K \rightarrow K')$. Of course, the above premise requires that the set S does not contain contradictions i.e. it does not contain both $Add(t)$ and $Del(t)$ for a given t. This is true for the comparison functions Δ_e, Δ_c, Δ_d, Δ_{dc}. However, this is not the only technical problem we have to address.

The order of execution of the change operations may affect the resulting KB, in particular the resulting KB may not satisfy all change operations returned by a comparison function (see Def. 3). For instance, for the KBs of Table 2 (d) we get $\Delta_{dc}(K' \rightarrow K) = \{Del(A \textit{ subClassOf } D), Del(B \textit{ subClassOf } D), Del(C \textit{ subClassOf } D)\}$. If the operations are executed in the order $\langle Del(A \textit{ subClassOf } D), Del(B \textit{ subClassOf } D), Del(C \textit{ subClassOf } D)\rangle$ under $\mathcal{U}_{ir}$ semantics, then all of them will be satisfied and the result will be equivalent to K. Now consider the following execution order $\langle Del(B \textit{ subClassOf } D), Del(C \textit{ subClassOf } D), Del(A \textit{ subClassOf } D)\rangle$. In this case the operation $Del(B \textit{ subClassOf } D)$ does not change the K as it requests the deletion of an inferred triple and according to $\mathcal{U}_{ir}$ semantics an inferred triple can't be deleted. The same will happen with the operation $Del(C \textit{ subClassOf } D)$. Finally, the operation $Del(A \textit{ subClassOf } D)$ will be executed and will cause the addition of the triple $(B \textit{ subClassOf } D)$. It is obvious that the operation $Del(B \textit{ subClassOf } D)$ is not satisfied by the resulting KB because it contains the triple $(B \textit{ subClassOf } D)$ i.e. $\Delta_{dc}(K \rightarrow K')^{\mathcal{U}_{ir}} = \mathcal{E}$. We have just seen an example where the order of execution matters. The same problem occurs when K' contains a redundant triple e.g. $(B \textit{ subClassOf } D)$. A similar situation is encountered with Δ_c and with Δ_d when K and K' are not redundancy free.

To avoid nondeterminism and to ensure correctness, we need an execution semantics of change operations (comprised in Deltas) that guarantees their satisfaction (if this is

possible). This can be achieved by: (a) defining comparison functions that return *sequences* (not sets) of change operations which guarantee satisfaction of their elements, or by (b) using a multi-pass execution mode that guarantees that all change operations will eventually be satisfied. Below we elaborate on the (b) approach. We could use a loop-based algorithm which terminates when every operation returned by a comparison function is satisfied.

Alg 1. **Execute**(K, M, sem) where $M \subseteq \mathcal{S}$, $sem \in \{\mathcal{U}_p, \mathcal{U}_{ir}\}$
(1) repeat
(2) get an element $u \in M$ that is not satisfied by K
(3) $K_t = u^{sem}(K)$ // i.e. apply on K the appropriate post-conditions of u wrt sem
(4) $K = K_t$
(5) until $\{u \in M | u \text{ not satisfied by } K\} = \emptyset$

In this context, we have to prove that the execution algorithm always terminates (if M has been derived from one of $\Delta_e,\ \Delta_c,\ \Delta_d,\ \Delta_{dc}$). It is clear that the loop always terminates for the case of $(\Delta_e, \mathcal{U}_p)$ because it yields operations that are always satisfiable. So we only have to study Δ_d, Δ_c and Δ_{dc} underg $\mathcal{U}_{ir}$-semantics.

Let Y be the satisfiable deletions and Z the unsatisfiable deletions at any point during the execution of the algorithm. If we prove that whenever $|Y| = 0$ we also have $|Z| = 0$ then we prove that our algorithm always terminates since all elements of M are satisfied.

Both Δ_c and Δ_{dc} produce the following set of delete statements: $X = \{\ Del(t') \mid t' \in C(K) - C(K')\ \}$. An element $Del(t)$ will be satisfied if $t \in R(K)$. So the set Y, i.e. the satisfiable deletions of X, is defined as $Y = R(K) \cap (C(K) - C(K')) = R(K) - C(K')$. Let's now define Z. Recall that a $Del(t')$, may not be satisfied (when applied to K) only if $t' \in C(K) - R(K)$. So the set Z, i.e. the unsatisfiable deletions of X, is defined as $Z = (C(K) - R(K)) \cap (C(K) - C(K')) = C(K) - (R(K) \cup C(K'))$.

Let's now investigate whether $|Y| = 0 \Rightarrow |Z| = 0$ holds. At first, notice that $Y = \emptyset \Leftrightarrow R(K) - C(K') = \emptyset \Leftrightarrow R(K) \subseteq C(K')$. Also note that $R(K) \subseteq C(K') \Rightarrow C(K) \subseteq C(K')$. This is based on the properties of the closure operator: if we have two sets A and B such that $A \subseteq B$ and B is closed with respect to the closure operator C (i.e. $C(B) = B$), then $C(A) \subseteq B$.

Returning to our problem, if $Y = \emptyset$ (that is if $R(K) \subseteq C(K')$), then the formula $Z = C(K) - (R(K) \cup C(K'))$ is equivalent to $Z = C(K) - C(K')$. But above we have seen that $Y = \emptyset \Rightarrow C(K) \subseteq C(K')$ too. It follows that $Z = \emptyset$. So the algorithm always terminates.

The above is actually the proof of the proposition: If $|R(K) - C(K')| = 0$ then $|C(K) - (R(K) \cup C(K'))| = 0$.

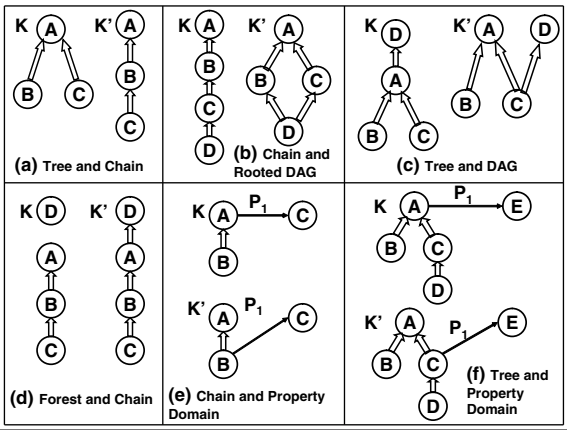
The proof for Δ_d is similar.

5.3 Identifying the Correct $(\Delta_x, \mathcal{U}_y)$-pairs

For identifying the pairs that are correct, Table 2 depicts 6 examples. For each example, it shows the result of applying Δ_d, Δ_c, Δ_e and Δ_{dc} for both $K \rightarrow K'$ and $K' \rightarrow K$, and contains the following columns:

- $\mathcal{U}_p$ Co: If Y then this means that $\Delta_x(K \rightarrow K')^{\mathcal{U}_p}(K) \sim K'$, i.e. the approach is *correct*. Otherwise the cell is marked with N.

Table 2. Examples



Delta	$K \rightarrow K'$	$\mathcal{U}_p$ Co	$\mathcal{U}_p$ RF	$\mathcal{U}_{ir}$ Co	$K' \rightarrow K$	$\mathcal{U}_p$ Co	$\mathcal{U}_p$ RF	$\mathcal{U}_{ir}$ Co
(a) Tree and Chain								
Δ_e	$\{Add(C\ subClassOf\ B), Del(C\ subClassOf\ A)\}$	Y	Y	Y	$\{Add(C\ subClassOf\ A), Del(C\ subClassOf\ B)\}$	Y	Y	Y
Δ_c	$\{Add(C\ subClassOf\ B)\}$	Y	N	Y	$\{Del(C\ subClassOf\ B)\}$	**N**	Y	Y
Δ_d	$\{Add(C\ subClassOf\ B)\}$	Y	N	Y	$\{Del(C\ subClassOf\ B)\}$	**N**	Y	Y
Δ_{dc}	$\{Add(C\ subClassOf\ B)\}$	Y	N	Y	$\{Del(C\ subClassOf\ B)\}$	**N**	Y	Y
(b) Chain and Rooted DAG								
Δ_e	$\{Add(C\ subClassOf\ A), Add(D\ subClassOf\ B), Del(C\ subClassOf\ B)\}$	Y	Y	Y	$\{Add(C\ subClassOf\ B), Del(C\ subClassOf\ A), Del(D\ subClassOf\ B)\}$	Y	Y	Y
Δ_c	$\{Del(C\ subClassOf\ B)\}$	**N**	Y	Y	$\{Add(C\ subClassOf\ B)\}$	Y	N	Y
Δ_d	$\{Del(C\ subClassOf\ B)\}$	**N**	Y	Y	$\{Add(C\ subClassOf\ B)\}$	Y	N	Y
Δ_{dc}	$\{Del(C\ subClassOf\ B)\}$	**N**	Y	Y	$\{Add(C\ subClassOf\ B)\}$	Y	N	Y
(c) Tree and DAG								
Δ_e	$\{Add(C\ subClassOf\ D), Del(A\ subClassOf\ D)\}$	Y	Y	**N**	$\{Add(A\ subClassOf\ D), Del(C\ subClassOf\ D)\}$	Y	Y	Y
Δ_c	$\{Del(A\ subClassOf\ D), Del(B\ subClassOf\ D)\}$	**N**	Y	Y	$\{Add(A\ subClassOf\ D), Add(B\ subClassOf\ D)\}$	Y	N	Y
Δ_d	$\{Del(A\ subClassOf\ D)\}$	**N**	Y	**N**	$\{Add(A\ subClassOf\ D)\}$	Y	N	Y
Δ_{dc}	$\{Del(A\ subClassOf\ D), Del(B\ subClassOf\ D)\}$	**N**	Y	Y	$\{Add(A\ subClassOf\ D)\}$	Y	N	Y
(d) Forest and Chain								
Δ_e	$\{Add(A\ subClassOf\ D)\}$	Y	Y	Y	$\{Del(A\ subClassOf\ D)\}$	Y	Y	**N**
Δ_c	$\{Add(A\ subClassOf\ D), Add(B\ subClassOf\ D), Add(C\ subClassOf\ D)\}$	Y	N	Y	$\{Del(A\ subClassOf\ D), Del(B\ subClassOf\ D), Del(C\ subClassOf\ D)\}$	Y	Y	Y
Δ_d	$\{Add(A\ subClassOf\ D)\}$	Y	Y	Y	$\{Del(A\ subClassOf\ D)\}$	Y	Y	**N**
Δ_{dc}	$\{Add(A\ subClassOf\ D)\}$	Y	Y	Y	$\{Del(A\ subClassOf\ D), Del(B\ subClassOf\ D), Del(C\ subClassOf\ D)\}$	Y	Y	Y
(e) Chain and Property Domain								
Δ_e	$\{Add(B, P_1, C), Del(A, P_1, C)\}$	Y	Y	Y	$\{Add(A, P_1, C), Del(B, P_1, C)\}$	Y	Y	Y
Δ_c	$\{Del(A, P_1, C)\}$	**N**	Y	Y	$\{Add(A, P_1, C)\}$	Y	N	Y
Δ_d	$\{Del(A, P_1, C)\}$	**N**	Y	Y	$\{Add(A, P_1, C)\}$	Y	N	Y
Δ_{dc}	$\{Del(A, P_1, C)\}$	**N**	Y	Y	$\{Add(A, P_1, C)\}$	Y	N	Y
(f) Tree and Property Domain								
Δ_e	$\{Add(C, P_1, E), Del(A, P_1, E)\}$	Y	Y	**N**	$\{Add(A, P_1, E), Del(C, P_1, E)\}$	Y	Y	Y
Δ_c	$\{Del(A, P_1, E), Del(B, P_1, E)\}$	**N**	Y	Y	$\{Add(A, P_1, E), Add(B, P_1, E)\}$	Y	N	Y
Δ_d	$\{Del(A, P_1, E)\}$	**N**	Y	**N**	$\{Add(A, P_1, E)\}$	Y	N	Y
Δ_{dc}	$\{Del(A, P_1, E), Del(B, P_1, E)\}$	**N**	Y	Y	$\{Add(A, P_1, E)\}$	Y	N	Y

- $\mathcal{U}_p$ RF: If Y then this means that the application of these changes results in a *redundancy free* K. Formally Y iff $RF(\Delta_x(K \rightarrow K')^{\mathcal{U}_p}(K)))$. Otherwise the cell is marked with N.
- $\mathcal{U}_{ir}$ Co: If Y then this means that $\Delta_x(K \rightarrow K')^{\mathcal{U}_{ir}}(K) \sim K'$ i.e the approach is correct. Otherwise the cell is marked with N.

In all cases we assume that the KBs are redundancy free. We do not have a column "$\mathcal{U}_{ir}$ RF" because by definition the execution of a $\mathcal{U}_{ir}$-operation leaves the knowledge base in a redundancy free state. Those pairs that have a N in the cells that concern correctness, constitute a proof (by counterexample) that they are *not correct*. For the rest pairs (those with a Y) we have to prove that they are *always* correct.

Theorem 1. For any pair of valid knowledge bases $\{K, K'\} \subseteq \Psi$ it holds:

$$\Delta_c(K \rightarrow K')^{\mathcal{U}_{ir}}(K) \sim \Delta_e(K \rightarrow K')^{\mathcal{U}_p}(K) \sim \Delta_{dc}(K \rightarrow K')^{\mathcal{U}_{ir}}(K) \sim K'$$

Theorem 2. $\Delta_d(K \rightarrow K')^{\mathcal{U}_{ir}}(K) \sim K'$ iff $\{K, K'\} \subseteq \Psi$ and either: (a) K is complete, or (b) $C(K) - K \subseteq C(K')$.

Due to space limitations the proof of the above theorems is omitted. An interesting remark regarding Th. 2 is that if $C(K') \supseteq C(K)$, then condition (b) holds. This means that we could use the pair $(\Delta_d, \mathcal{U}_{ir})$ in cases we know that $C(K') \supseteq C(K)$. For example if K is an ontology O and K' is an additional ontology O' that specializes O, then we are sure that $C(K') \supseteq C(K)$. In such cases we can use Δ_d (or alternatively Δ_{dc}) which give the smallest in size Deltas (Δ_{dc} returns the same Deltas).

5.4 Semantic Identify and Non Redundancy Properties of $(\Delta_x, \mathcal{U}_y)$-Pairs

Prop. 2. If $K \sim K'$ then $\Delta_d(K \rightarrow K') = \Delta_c(K \rightarrow K') = \Delta_{dc}(K \rightarrow K') = \emptyset$.

This is property (P1) and its proof is trivial. Note that Δ_e is not included in Prop. 2 because even if $K \sim K'$, it may be $K = K'$, $K \subset K'$, $K' \subset K$, or $K \not\subseteq K'$ and $K' \not\subseteq K$. In the example of Figure 3 (a) we get $\Delta_e(K \rightarrow K') = \{Add(C \textit{ subClassOf } A)\}$ although $K \sim K'$. It should be stressed that most of the existing comparison functions [3,31,10] actually employ Δ_e, so they do not satisfy (P1).

However, one can easily prove that: If K and K' are both redundancy free, and the knowledge bases considered have always a unique reduction, then $K \sim K' \Rightarrow \Delta_e(K \rightarrow K') = \emptyset$. In general, if the transitive closure of a binary relation R is antisymmetric and finite, then the transitive reduction of R is unique. In the problem at

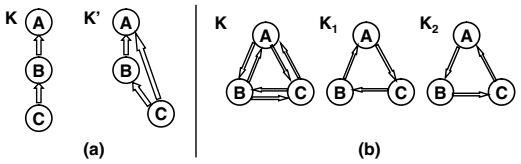


Fig. 3. K, K'

hand, if an RDF knowledge base allows forming cycles with subsumption relationships, then the transitive reduction is not unique. For example, in Figure 3 (b) we have $K \sim K_1 \sim K_2$, moreover $RF(K_1), RF(K_2)$, but $K_1 \neq K_2$.

Prop. 3. If $K \sim K'$, $\{K, K'\} \subseteq \Psi$ and $RF(K), RF(K')$ then $\Delta_e(K \rightarrow K') = \emptyset$

5.5 Summarizing the Results

The pairs that are always correct are: $(\Delta_c, \mathcal{U}_{ir})$, $(\Delta_e, \mathcal{U}_p)$ and $(\Delta_{dc}, \mathcal{U}_{ir})$. The pair $(\Delta_c, \mathcal{U}_p)$ is correct if K is complete. The pair $(\Delta_d, \mathcal{U}_{ir})$ is correct in the cases specified in Theorem 2. The set of change operations derived from either Δ_c or Δ_{dc} need the multi-pass execution mode while Δ_e requires a single pass execution mode. Concerning the size criterion, Δ_d produces the smallest in size result. Δ_{dc} produces smaller results than Δ_c. Concluding, we can say that the pairs $(\Delta_{dc}, \mathcal{U}_{ir})$ and $(\Delta_e, \mathcal{U}_p)$ are the *most appropriate for implementing change-based versioning services*: they are always correct and the size of Δ_{dc} is less than Δ_c. We cannot however compare the size of Δ_e with that of Δ_{dc} (in some cases the first is smaller, in others the second). Table 3 synopsizes the results. Concerning the column labeled "Execution Mode", S is used to denote single pass, and M to denote multi pass.

It is worth mentioning that $(\Delta_e, \mathcal{U}_p)$ is correct even if we are not in Ψ. Moreover, Theorem 1 holds even if we are not in Ψ but we adopt a "batch" execution mode for $\mathcal{U}_{ir}$, where each change operation is not executed independently but all change operations of the produced Delta are executed as one "transaction", i.e. we compute the closure and the reduction only once.

Table 3. Synopsis

Comp.	Sem.	Always Correct	Exec Mode	(P1)	(P2)	(P2.1)
Δ_e	$\mathcal{U}_p$	**Y**	S	see (Prop. 3)	N	Y
Δ_c	$\mathcal{U}_p$	**Y** if K complete	S	Y	N	
Δ_d	$\mathcal{U}_p$	N		Y	N	
Δ_{dc}	$\mathcal{U}_p$	N		Y	N	
Δ_e	$\mathcal{U}_{ir}$	N		see (Prop. 3)	Y	Y
Δ_c	$\mathcal{U}_{ir}$	**Y** in Ψ	M	Y	Y	Y
Δ_d	$\mathcal{U}_{ir}$	(see Theorem 2)	M	Y	Y	Y
Δ_{dc}	$\mathcal{U}_{ir}$	**Y** in Ψ	M	Y	Y	Y

6 Concluding Remarks

One approach for computing the difference between two RDF models is to take the difference between the sets of triples forming the two models (along with some refinements such as taking into account blank nodes). Another approach (useful for

versioning) is to identify a set of change operations that will transform one model into the other. In this paper we investigated the second approach and studied different semantics for this computation as well as properties like minimality and correctness of the produced Deltas. Most of the existing RDF comparison tools [3,31,10] rely on the $(\Delta_e, \mathcal{U}_p)$ pair. Semversion [31] offers also $(\Delta_c, \mathcal{U}_p)$ for the case where the K is complete (we have proved that in such cases this approach yields correct results). None of the works (theoretical or practical) has used Δ_d or Δ_{dc}. Recall that we have shown that $(\Delta_{dc}, \mathcal{U}_{ir})$ is better than $(\Delta_c, \mathcal{U}_p)$ not only because $(\Delta_{dc}, \mathcal{U}_{ir})$ does not require the KBs to be complete, but also because it returns smaller in size Deltas. We have identified the cases where $(\Delta_d, \mathcal{U}_{ir})$ is beneficial (recall that Δ_d gives the minimum in size Deltas). An issue for further research is to identify the conditions under which Δ_e yields smaller Deltas than Δ_{dc} and vice versa.

In comparison with belief contraction-revision (e.g. [14,20,11]), these theories consider KBs as logic theories and focus on what the result of applying a contraction/ revision operation on a KB should be. In our setting, the destination KB is known, i.e. it is K', so the focus is given on the transition from K to K'[6].

We plan to exploit the properties of the various Delta functions presented in this paper for building versioning services on top of SW repositories [28]. For reasons of space, technical details, as well as issues regarding the peculiarities of RDF including blank nodes identification, and containers (Bag, Sequence, Alternative) are omitted[7]. An important implementation issue that is worth mentioning is that the algorithm implementing the Delta functions never computes the closure of a KB. Instead, it constructs the explicit graph of each KB, and then it checks whether $t \in C(K)$ which can be decided efficiently (in $O(1)$), thanks to a labeling scheme [7] for subsumption relationships that is supported by RDFSuite [12]. Concerning the execution of $\mathcal{U}_{ir}$ operations, related algorithms include [26], while a similar in spirit approach for RDF has already been implemented for the RUL language [21].

Acknowledgements. This work was partially supported by the EU projects CASPAR (FP6-2005-IST-033572) and KP-Lab (FP6-2004-IST-4).

References

1. Abiteboul, S., Vianu, V.: Equivalence and optimization of relational transactions. Journal of the ACM (JACM) 35(1), 70–120 (1988)
2. Alchourrón, C.E., Gärdenfors, P., Makinson, D.: On the Logic of Theory Change: Partial Meet Contraction and Revision Functions. J. Symb. Log. 50(2), 510–530 (1985)
3. Beners-Lee, T., Connoly, D.: Delta: An Ontology for the Distribution of Differences Between RDF Graphs, (version: 2006-05-12) (2004), `http://www.w3.org/DesignIssues/Diff`

[6] Note that $\mathcal{U}_p, \mathcal{U}_{ir}$ are not proposed as general purpose change operations, but only for executing the results of the comparison functions we have defined in this paper.

[7] One approach to tackle the blank node identification problem is to consider that a KB is not a set of RDF triples but a set of RDF molecules [10,13,27]. Alternative techniques include [6].

4. Berliner, B.: CVS II: Parallelizing Software Development. In: Procs of the USENIX Winter 1990 Technical Conference, Berkeley, CA, pp. 341–352 (1990), citeseer.ist.psu.edu/berliner90cvs.html
5. Brickley, D. and Guha, R.V. RDF Vocabulary Description Language 1.0: RDF Schema, W3C Recommendation (February 2004) http://www.w3.org/TR/rdf-schema/
6. Carroll, J.J.: Matching RDF graphs. In: Horrocks, I., Hendler, J. (eds.) ISWC 2002. LNCS, vol. 2342, pp. 5–15. Springer, Heidelberg (2002)
7. Christophides, V., Plexousakis, D., Scholl, M., Tourtounis, S.: On Labeling Schemes for the Semantic Web. In: Procs. of WWW 2003, Budapest, Hungary, pp. 544–555 (May 2003)
8. Cloran, R., Irwin, B.: Transmitting RDF graph deltas for a Cheaper Semantic Web. In: Procs. of SATNAC 2005. South Africa (September 2005)
9. Conradi, R., Westfechtel, B.: Version models for software configuration management. ACM Comput. Surv. 30(2), 232–282 (1998)
10. Ding, L., Finin, T., Joshi, A., Peng, Y., Silva, P.d., McGuinness, D.: Tracking RDF Graph Provenance using RDF Molecules. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, Springer, Heidelberg (2005)
11. Flouris, G.: On Belief Change and Ontology Evolution. PhD thesis, Computer Science Department, University of Crete, Greece (2006)
12. FORTH-ICS. The ICS-FORTH RDFSuite: High-level Scalable Tools for the Semantic Web (2005), http://139.91.183.30:9090/RDF/
13. Petersson, J., Piazza, F., Puliti, P., Tummarello, G., Morbidoni, C.: RDFGrowth, a P2P annotation exchange algorithm for scalable Semantic Web applications. In: 1st Annual International Conference on Mobile and Ubiquitous Systems MobiQuitous, Boston, MA, August 2004 (2004)
14. Gärdenfors, P.: Belief Revision: An Introduction. In: Belief Revision, pp. 1–20. Cambridge University Press, Cambridge (1992)
15. Gutierrez, C., Hurtado, C., Mendelzon, A.: Foundations of Semantic Web Databases. In: PODS 2004. 23 ACM Symposium on Principles of Database Systems, ACM Press, New York (2004)
16. Hayes, P. RDF Semantics, W3C Recommendation (February 2004) http://www.w3.org/TR/rdf-mt/
17. Heflin, J., Hendler, J., Luke, S.: Coping with Changing Ontologies in a Distributed Environment. In: Procs. of AAAI 1999 Workshop on Ontology Management, Florida (July 1999)
18. Klein, M., Fensel, D., Kiryakov, A., Ognyanov, D.: Ontology versioning and change detection on the web. In: Gómez-Pérez, A., Benjamins, V.R. (eds.) EKAW 2002. LNCS (LNAI), vol. 2473, pp. 197–212. Springer, Heidelberg (2002)
19. Klein, M., Noy, N.: A component-based framework for ontology evolution. In: IJCAI 2003. Workshop on Ontologies and Distributed Systems, Acapulco, Mexico (2003)
20. Konieczny, S., Perez, R.P.: Propositional Belief Base Merging or How to Merge Beliefs/Goals Coming from Several Sources and Some Links With Social Choice Theory. European Journal of Operational Research 160(3), 785–802 (2005)
21. Magiridou, M., Sahtouris, S., Christophides, V., Koubarakis, M.: RUL:A Declarative Update Language for RDF. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 506–521. Springer, Heidelberg (2005)
22. Klein, M., Noy, N.F., Kunnatur, S., Musen, M.A.: Tracking Changes During Ontology Evolution. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 259–273. Springer, Heidelberg (2004)
23. Noy, N.F., Musen, M.A.: Prompt Diff: A Fixed-point Algorithm for Comparing Ontology Versions. In: Procs of AAAI 2002, Edmonton, Alberta, pp. 744–750 (July 2002)
24. Noy, N.F., Musen, M.A.: Ontology versioning in an ontology management framework. IEEE Intelligent Systems 19(4), 6–13 (2004)

25. Plessers, P., Troyer, O.D.: Ontology Change Detection Using a Version Log. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 578–592. Springer, Heidelberg (2005)
26. La Poutre, J.A., van Leeuwen, J.: Maintenance of Transitive Closures and Transitive Reductions of Graphs. In: Procs. of the Intern. Workshop on Graph-Theoretic Concepts in Computer Science, pp. 106–120 (July 1987)
27. Stickler, P: CBD - Concise Bounded Description. W3C Member Submission June 2005), http://www.w3.org/Submission/CBD/
28. Theoharis, Y., Christophides, V., Karvounarakis, G.: Benchmarking Database Representations of RDF/S Stores. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 685–701. Springer, Heidelberg (2005)
29. Tichy, W.F.: RCS-a system for version control. Software Practice & Experience 15(7), 637–654 (1985)
30. Tzitzikas, Y., Kotzinos, D. (Semantic Web) Evolution through Change Logs: Problems and Solutions. In: Procs of AIA 2007, Innsbruck, Austria (February 2007)
31. Volkel, M., Winkler, W., Sure, Y., Kruk, S.R., Synak, M.: SemVersion: A Versioning System for RDF and Ontologies. In: Gómez-Pérez, A., Euzenat, J. (eds.) ESWC 2005. LNCS, vol. 3532, Springer, Heidelberg (2005)
32. Zhang, Z., Zhang, L., Lin, C., Zhao, Y., Yu, Y.: Data Migration for Ontology Evolution. In: Fensel, D., Sycara, K.P., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, Springer, Heidelberg (2003)