J Data Semant (2014) 3:75–106
DOI 10.1007/s13740-013-0027-z

ORIGINAL ARTICLE

Curating the Specificity of Ontological Descriptions under Ontology Evolution

Yannis Tzitzikas · Mary Kampouraki · Anastasia Analyti

Received: 29 October 2012 / Revised: 22 April 2013 / Accepted: 30 May 2013 / Published online: 21 June 2013

Abstract Semantic Web Ontologies are not static, but evolve, however, this evolution usually happens independently of the ontological instance descriptions (usually referred as "metadata") which are stored in the various Metadata Repositories or Knowledge Bases. Nevertheless, it is a common practice for a MR/KB to periodically update its ontologies to their latest versions by "migrating" the available instance descriptions to the latest ontology versions. Such migrations incur gaps regarding the specificity of the migrated metadata, i.e. inability to distinguish those descriptions that should be reexamined (for possible specialization as consequence of the migration) from those for which no reexamination is justified. Consequently, there is a need for principles, techniques, and tools for managing the uncertainty incurred by such migrations, specifically techniques for (a) identifying automatically the descriptions that are candidates for specialization, (b) computing, ranking and recommending possible specializations, and (c) flexible interactive techniques for updating the available descriptions (and their candidate specializations), after the user (curator of the repository) accepts/rejects such recommendations. This problem is especially important for curated knowledge bases which have increased quality requirements (as in E-science). In this paper, we formalize the problem and propose principles, rules, and algorithms for realizing the aforementioned scenario, for the RDF/S framework. Finally, we report experimental results demonstrating the feasibility of the approach.

Keywords Ontology evolution · Quality (uncertainty) management in curated KBs · Semantic web data and ontologies

Electronic supplementary material The online version of this article (doi:10.1007/s13740-013-0027-z) contains supplementary material, which is available to authorized users.

Y. Tzitzikas · M. Kampouraki · A. Analyti (✉)
Institute of Computer Science, FORTH-ICS, Crete, Greece
e-mail: analyti@ics.forth.gr

Y. Tzitzikas
e-mail: tzitzik@ics.forth.gr

M. Kampouraki
e-mail: mary.kampouraki@gmail.com

Y. Tzitzikas
Department of Computer Science, University of Crete, Crete, Greece

1 Introduction

Ontologies provide a shared conceptualization of a domain by defining the concepts in the domain and describing how those concepts are related to each other. Several reasons for changing an ontology have been identified in the literature [9]: an ontology may need to change because it offers a richer conceptualization of the problem domain, the domain of interest has been changed, the perspective under which the domain is viewed has changed, or the user/application needs have changed.

An important observation is that this evolution happens *independently* of the ontological instance descriptions which are stored in the various Metadata Repositories (MRs) or Knowledge Bases (KBs). With the term *ontological instance description*, we refer to RDF/S [3] descriptions that classify an instance o to a class c or relate two instances o, o' with a property pr. With the term MR or KB, we refer to a stored corpus of ontological instance descriptions, which can be stored in files, in RDF/S databases (i.e. RDF triple-stores [18]), or in the rapidly growing *Linked Open Data* (LOD) cloud [2]. Due to the distributed nature of the Web and the Semantic

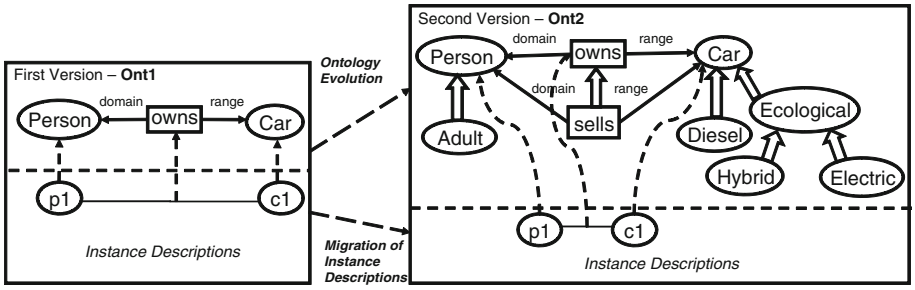

Fig. 1 Introductory example

Web, the evolution of ontologies[1] happens independently of the ontological instance descriptions, e.g. this is the case with ontologies maintained by standardization authorities. However, it is a common practice (mainly for interoperability purposes) for a KB to periodically update its ontologies to their latest versions by "migrating" the stored instance descriptions to the latest ontology versions. Such migrations are in many cases not difficult, because newer versions are (or constructed to be) compatible with past ones. Nevertheless, they incur gaps regarding the specificity of the migrated instance descriptions, i.e. inability to distinguish those that should be reexamined (for possible specialization as consequence of the migration) from those for which no reexamination is justified. It follows that quality control is very laborious and error-prone.

To start with, consider a corpus of instance descriptions and suppose that at certain points in time we can make the assumption that the available instance descriptions are the *most specific and detailed descriptions* that are possible with respect to the employed ontology. For instance, we can make such an assumption after explicit human (e.g. by the curator of the KB) inspection and verification [4], or in cases where the descriptions have been produced automatically by a method that is guaranteed to produce specific descriptions (e.g. by transforming curated relational data to RDF/S descriptions [25], or by automatic classification to categories each defined by sufficient and necessary conditions, etc.). We will hereafter refer to this assumption by the name *maximum specificity assumption* (for short *MSA*). It is not hard to see that if the new version of the ontology is *richer* than the past one, then the corpus of the migrated instance descriptions *may no longer satisfy the MSA with respect to the new ontology*. This paper elaborates on this problem.

The ability to identify the instance descriptions that satisfy the *MSA* and those that do not, is useful in order to address questions of the form: (a) for what descriptions can we make the *MSA*? (b) what (class or property) instances should probably be reclassified (to more refined classes or properties), and (c) which are the candidate new classes or properties (refinements) of such instances? The above questions are very useful for curating a corpus of instance descriptions, i.e. for managing its specificity as the corpus (and its ontologies) evolves over time. Without special support, such tasks would be unacceptably expensive and vulnerable to omissions, for large datasets. The problem occurs in various domains, including Digital Libraries (e.g. as the *Library of Congress Subject Headings LCSH* evolves), in Biomedicine/Bioinformatics (*Gene Ontology*), in e-Government (*oeGOV Ontologies*), etc.

Introductory (toy) example. We will explain the main idea of our approach using the toy example depicted at Fig. 1. Consider an e-commerce portal that sells various kinds of products. Suppose a car `c1` that has been classified under the class `Car`, and a person `p1` that has been classified under the class `Person`, defined in an ontology `Ont1`, and suppose that both classes have no subclasses. Assume that for the current set of instance descriptions according to `Ont1` the *MSA* holds (i.e. they are complete with respect to specificity). We can infer, from this knowledge, that `c1` *is not* a `Person` and `p1` *is not* a `Car`. Let `Ont2` be a new version of that ontology which, among other, defines the subclasses of the classes `Car` and `Person`, shown at Fig. 1 (right). All subclasses of `Car` are *possible classes* for `c1`. `Adult` *is not* a possible class for `c1`, since `c1` was not a person according to `Ont1`. Analogously, none of the subclasses of `Car` is a possible class for `p1`, since `p1` was not a car according to `Ont1`. Moreover, notice that `Ont1` defines a property `owns` and suppose that (`p1 owns c1`) is an instance description. Also notice that `Ont2` defines a subproperty `sells` of `owns` between `Person` and `Car`. This property will be prompted as a *possible specialization of the association* between `p1` and `c1`.

The Difficulties. In real cases the computation of possibilities is more complex than the case of the toy example, as we can have conflicts among (a) new positive knowledge inferable from the instance descriptions and the new schema, (b) new "negative" information inferable from the past negative instance descriptions and the new schema, and (c) the previously computed possible instance descriptions (possible refinements). In fact, our approach resolves such conflicts by considering that (a) has higher priority than (b), and (b) has higher priority than (c). Our resolution of conflicts is illustrated in Figs. 2 and 3.

[1] In this paper, by the term of ontology we refer only to schema information.

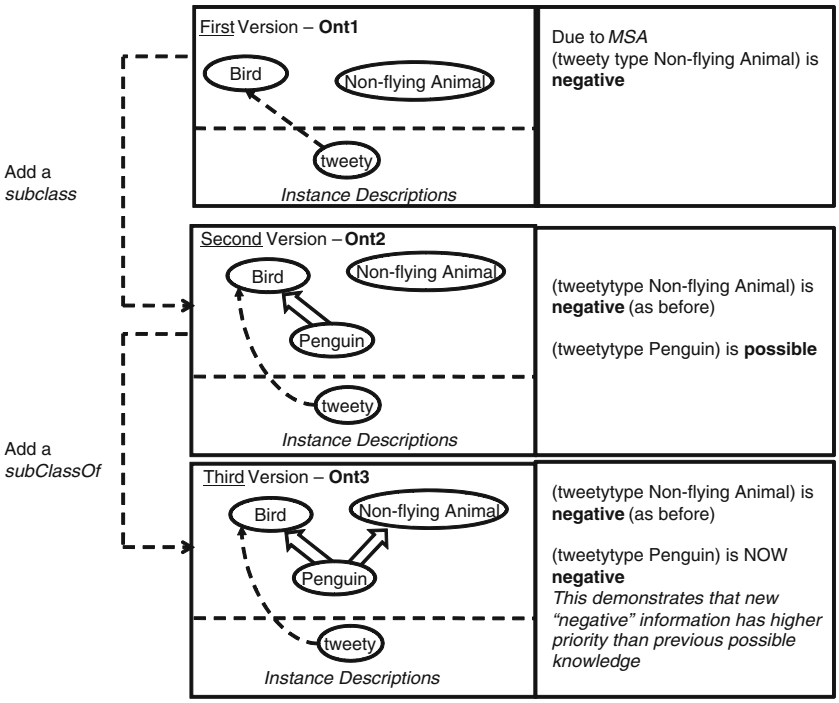

Fig. 2 New "negative" information has higher priority than previous possible knowledge

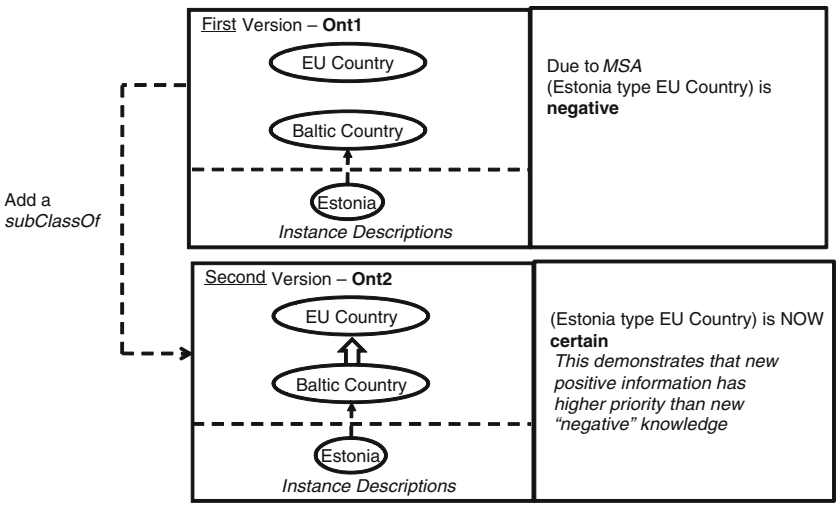

Fig. 3 New positive information has higher priority than the new "negative" knowledge

In addition, it should be possible to update correctly the set of possibilities, at scenarios with several *successive* instance migrations interwoven with several (positive or negative) user feedbacks. Finally, another challenge is to reduce the information that has to be kept to support this scenario, specifically to avoid having to keep negative information of any kind, and to devise compact representations for the possibilities.

The contribution of this work lies in (a) formalizing the notion of *(possible) specificity*, (b) providing principles, rules, and algorithms for computing possibilities after schema migrations, (c) describing a flexible specificity-aware curation process, and (d) reporting experimental results (over real and synthetic datasets) that demonstrate the feasibility of the approach. We could say that from a more general perspective, our work contributes in enriching the lifecycle of Semantic Web data with quality management, appropriate for scenarios where ontologies evolve frequently and independently from instance descriptions. As a consequence, this allows adopting iterative and agile ontology modeling approaches, appropriate for open environments like Linked Open Data.

Note that though there are several works and approaches for dealing with the validity of data during migration [16, 19, 22], there is no work for managing their specificity and quality while ontologies evolve.

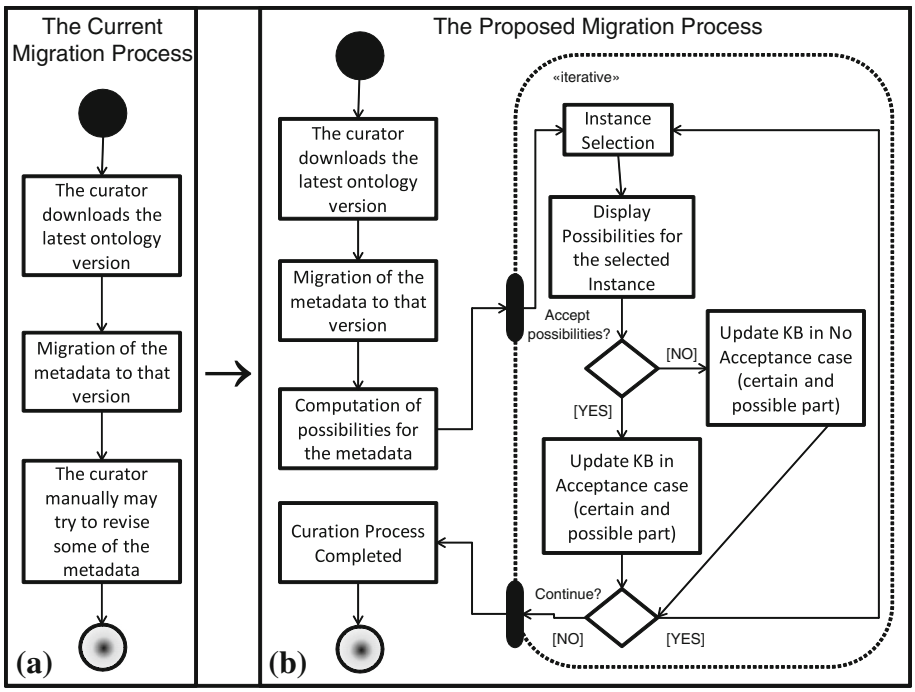

Fig. 4 Current and proposed migration process

Here, we would like to mention that an overview of the main principles of this paper (demonstrated through examples) and an overview of the functionality of the prototype system, all from the perspective of *digital preservation*, is given in our conference paper [29].

The rest of this paper is organized as follows. Section 2 gives the required background information and notations. Section 3 introduces what we call *TFP-partition* which is fundamental in our approach. Section 4 formalizes the migration problem as a *transition* between TFP-partitions, and proposes the basic *postulates* that should govern such transitions. Based on the above formalization and postulates, Sect. 5 provides an algorithm for computing the set of possible instance descriptions of an RDF/S KB after a migration, for the case of *backwards compatible* schema evolution, while Sect. 6 considers the same problem for the more general case of *non-backwards compatible* schema evolution. Section 7 discusses the effect of *successive* migrations. Section 8 presents methods for *ranking possibilities*. Section 9 describes the specificity lifecycle management process (for exploiting the computed possibilities). Section 10 provides experimental results over both real and synthetic data sets, and discusses storage approaches. Section 11 discusses related work. Finally, Sect. 12 concludes the paper and identifies issues for further research. Appendix A provides a list with the symbols used in the paper and Appendix B provides an example of applying the algorithm for backwards compatible migration. An extra electronic Appendix of this paper provides proofs and supplementary material.

2 Approach and Background

2.1 The Curation Process Model

Apart from identifying the information that could be further specialized (as we discussed just before), we would like to *aid* making it as specific as possible. Therefore, we should support flexible and *interactive* processes for managing the computed possibilities, where the user will be able to either *accept* or *reject* the computed *recommendations*, and eventually update the knowledge base reaching to a state where the *MSA* holds (at least for those resources for which this is possible). The *ranking* of possibilities is important for designing user-friendly interaction schemes, since we may have a high number of recommendations. Essentially, we propose a process like the one sketched in the right part of Fig. 4. Specifically, assume that the user selects some instances then the system displays ranked all or some of the possible instance descriptions for the selected instances. The user accepts or rejects these instance descriptions and the system updates appropriately the KB and its possible part. Note that the possible part of the KB is stored explicitly and separately. In our toy example, this means that we can rank the possible classes for `c1`, so that if the user is prompted to select a possible class for `c1`, then `Diesel` and `Ecological` will be the first classes to be displayed. If the user rejects the class `Ecological`, then all its subclasses will be rejected from the possible classes (and this reduces the effort required for reaching a state where the *MSA* holds).

2.2 On MSA, RDF, and Open/Closed World Assumptions

To avoid misunderstandings, we should clarify from the beginning that we do not violate the Open World Assumption of RDF/S. It is the *MSA* that allows us to infer the negative knowledge of the previous example. Only if we explicitly make the *MSA* we can then exploit it (in a Closed World Assumption manner) in the context of ontology evolution for formalizing the way possibilities are defined. To clarify that we can also capture the Open World Assumption of RDF, suppose that we start from an RDF/S KB for which we know nothing regarding its completeness or specificity. We can capture this case by considering that every instance description, that can be formed using the ontology and is not certain, is possible. That is, the set of "negative assertions" is empty (in our example that would mean that `Adult` *can be* a possible class for `c1` and `Car` *can be* a possible class for `p1`). Then, we can still use and exploit our machinery when we migrate our descriptions to subsequent schema versions, and the steps of the life cycle that we propose can produce the "negative" statements. If however one knows that one particular RDF/S KB is complete (regarding specificity), which can be true in the context of curated knowledge bases, then he can "apply" the *MSA*. This means that every instance description, that can be formed using the ontology and is not certain, is negative. Thus, the set of possibilities is empty (in our example that would mean that `Adult` *is not* a possible class for `c1` and `Car` *is not* a possible class for `p1`).

As we shall see later, the notion of TFP-partition allows modeling all such cases, and in Sect. 3.1 will show how exactly this can be done.

2.3 RDF/S Knowledge Bases

This section introduces notions and notations about RDF/S that shall be used in the sequel. Let *URI* be the set of URI references and *LIT* be the set of plain and typed literals. In our framework, an RDF/S Knowledge Base (KB) is defined by a set of RDF triples of the form (*subject predicate object*), where *subject*, *predicate* $\in$ *URI* and *object* $\in$ *URI* $\cup$ *LIT*.

Let $\mathcal{T}$ be the set of all possible triples that can be constructed from a countably infinite set of URIs, as well as literals (e.g. strings, integers, float numbers) [11]. Then, an RDF/S KB (for short 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 KB K, other triples can be inferred based on the RDF/S semantics [13]. For this reason, we introduce the notion of closure.

The *closure* of a KB K, denoted by $\mathcal{C}(K)$, is the set of all triples that either are explicitly asserted or can be inferred from K based on RDFS-entailment of the RDF/S semantics [13]. Essentially, the following derivation rules are used:

(i) if $(c_1\ subClassOf\ c_2)$ and $(c_2\ subClassOf\ c_3)$ then $(c_1\ subClassOf\ c_3)$,
(ii) if $(pr_1\ subPropertyOf\ pr_2)$ and $(pr_2\ subPropertyOf\ pr_3)$ then $(pr_1\ subPropertyOf\ pr_3)$,
(iii) if $(o\ type\ c_1)$ and $(c_1\ subClassOf\ c_2)$ then $(o\ type\ c_2)$,
(iv) if $(o\ \ pr_1\ \ o')$ and $(pr_1\ subPropertyOf\ pr_2)$ then $(o\ pr_2\ o')$,
(v) if $(o\ pr\ o')$ and $(pr\ domain\ c)$ then $(o\ type\ c)$, and
(vi) if $(o\ pr\ o')$ and $(pr\ range\ c)$ then $(o'\ type\ c)$.

Definition 1 Let K be a KB. We define the *schema* of K as the tuple $\Gamma_K = \langle C_K,\ Pr_K,\ domain, range,\ \leq^*_{cl},\ \leq^*_{pr}\rangle$, where[2]:

- C_K is the set of classes of $\mathcal{C}(K)$,
- Pr_K is the set of properties of $\mathcal{C}(K)$,
- $domain$ is a total function $domain:\ Pr_K \rightarrow C_K$ that maps a property in Pr_K to its domain,
- $range$ is a total function $range:\ Pr_K \rightarrow C_K$ that maps a property in Pr_K to its range,
- $\leq^*_{cl}$ is the *subClassOf* relation between C_K, and
- $\leq^*_{pr}$ is the *subPropertyOf* relation between Pr_K.

Below we introduce notations for the *resources* of K, the *instances* of K, and the *instances of a class* $c \in C_K$:

$$Res_K = \{o \mid (o\ type\ Resource) \in \mathcal{C}(K)\}$$
$$Inst_K = Res_K \setminus (C_K \cup Pr_K)$$
$$inst_K(c) = \{o \mid (o\ type\ c) \in \mathcal{C}(K)\}$$

Definition 2 (*Valid KB*) We consider a KB K to be *valid* if:

(i) the relations $\leq^*_{cl}$ and $\leq^*_{pr}$ are acyclic,
(ii) if $pr \leq^*_{pr} pr'$ then $domain(pr) \leq^*_{cl} domain(pr')$ and $range(pr) \leq^*_{cl} range(pr')$.

In this paper, we consider only valid KBs. The triples of $\mathcal{T}$ can be partitioned to *schema* and *instance* triples, as shown in Table 1. The triples that are not schema triples, according to that table, are instance triples. The latter can be further partitioned to *class instance triples* [having the form $(o\ type\ c)$] and *property instance triples* [having the form $(o\ pr\ o')$].

Example 1 Figure 5 illustrates two KBs K and K', each consisting of schema triples and instance triples. In this example the instances are shown at the bottom of each KB and we have $Inst_K = Inst_{K'} =$ {`Fiat_1`, `BMW_1`, `Bob`, `Alice`, `Computer Science Department`, `FORTH`}.

[2] Note that according to RDF/S semantics [13], $\leq^*_{cl}$ and $\leq^*_{pr}$ are reflexive and transitive relations.

Table 1 Schema and instance triples

Schema triples	
Triple	Abbreviation
c *type Class*	
c *subClassOf* c'	$c \leq_{cl} c'$
pr *type Property*	
pr *subPropertyOf* pr'	$pr \leq_{pr} pr'$
pr *domain* c''	$domain(pr) = c''$
pr *range* c''	$range(pr) = c''$
Instance triples	
o *type* c	
o pr o'	

3 The Notion of TFP-partition

In this Section, we introduce the notion of TFP-partition (True-False-Possible partition) of a KB for capturing various application-specific assumptions about the specificity of its descriptions. At first, we define the set of *cartesian instance triples* of a KB K, denoted by B_K (where "B" comes from Base). Intuitively, it includes all instance triples that can be expressed using the schema of K.

Definition 3 (*Cartesian Instance Triples*) Given a KB K, the set of *cartesian instance triples* of K, denoted by B_K, is the union of the class instance triples in $(Inst_K \cap URI) \times \{type\} \times C_K$ and the property instance triples in $(Inst_K \cap URI) \times Pr_K \times Inst_K$.

Given a KB K, we can distinguish its set of *schema triples* S_K and its set of *instance triples* I_K, i.e. $K = S_K \cup I_K$. However, for migration purposes, we need to consider only instance triples in B_K, i.e. those in $I_K \cap B_K$. Note that the remaining instance triples, e.g. property instances that connect classes, are not interesting for migration purposes, and we ignore them. For notational simplicity we shall hereafter use I_K to denote $I_K \cap B_K$, and we shall use $K = (S_K, I_K)$. We define $\mathcal{C}_i(K)$ as the set of explicit and inferred instance triples, specifically $\mathcal{C}_i(K) = \mathcal{C}(K) \cap B_K$. Clearly, it holds: $I_K \subseteq \mathcal{C}_i(K) \subseteq B_K$.

We shall write $K = (S_K, I_K)$. We define $\mathcal{C}_i(K)$ as the set of explicit and inferred instance triples, specifically $\mathcal{C}_i(K) = \mathcal{C}(K) \cap B_K$. Clearly, it holds: $I_K \subseteq \mathcal{C}_i(K) \subseteq B_K$.

In the Example 1, we have $K = (S_K, I_K)$, $K' = (S_{K'}, I_{K'})$ and $I_K = I_{K'}$, e.g. {(Bob drives BMW_1), (Fiat_1 type Car)} $\subset I_K = I_{K'}$. In addition we have $(domain(\texttt{drives}) = \texttt{Person}) \in S_K$, $(\texttt{Car} \leq^*_{cl} \texttt{Vehicle}) \in S_{K'}$, while $(\texttt{Car} \leq^*_{cl} \texttt{Vehicle}) \notin S_K$.

Below, we define the notion of *valid to add property instance* in order to indicate which of the property instance triples of B_K are considered invalid.

Definition 4 (*Valid to add property instance*) We shall call a property instance triple $(o\ pr\ o') \in B_K$ *valid to add*, for short *valid*, if it satisfies the constraint:

$$(o \in inst_K(domain(pr))) \wedge (o' \in inst_K(range(pr))) \quad (1)$$

The motivation is the following: if we add to K a property instance triple $(o\ pr\ o')$ that does not satisfy expression (1) of Definition 4, then at least one new class instance triple would be added to $\mathcal{C}_i(K)$, due to derivation rules (v) and

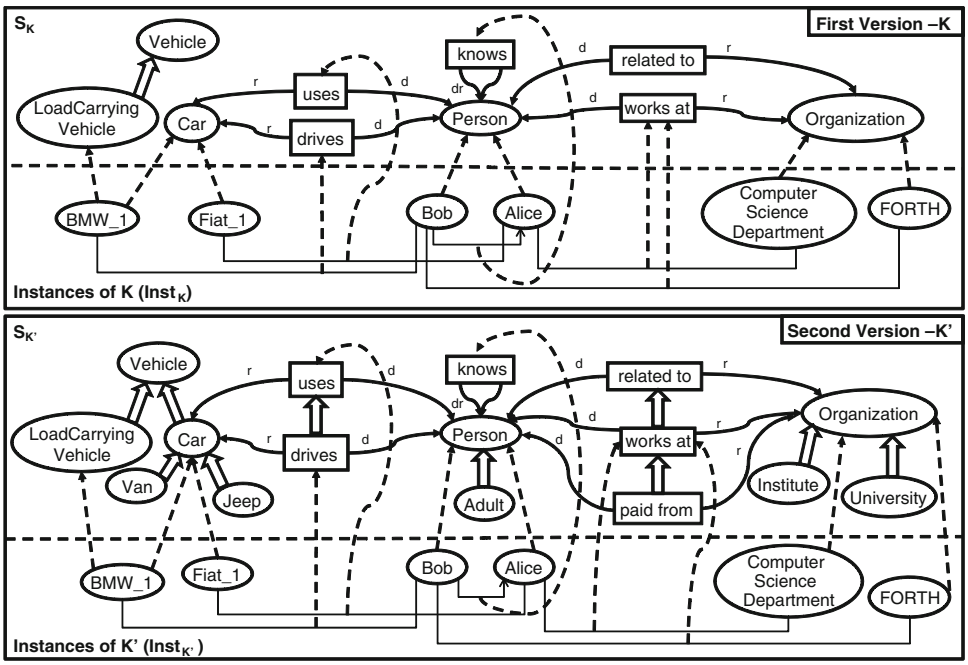

Fig. 5 An instance migration scenario Classes and class instances are depicted by *ovals*. Properties are depicted by *rectangles* and the letters "d" and "r" are used to denote the domain and the range of a property. *Fat arrows* denote subClassOf/subPropertyOf relationships, while *dashed arrows* denote instanceOf relationships

(vi) of RDF/S semantics (see Sect. 2). Furthermore, several Semantic Web Repositories for well justified reasons do not support derivation rules (v) and (vi), and consequently do not accept the addition of property instance triples that do not satisfy expression (1). To simplify notation, hereafter we shall use $valid(o, pr, o', K)$ to denote expression (1). We shall also use $Invalid(K)$ to refer to the invalid property instance triples of B_K, i.e.

$$Invalid(K) = \{(o\ pr\ o') \in B_K \mid \neg valid(o,\ pr,\ o',\ K)\}$$

Let us also introduce some auxiliary notations. We will define the $SubTriples$ of an instance triple as follows:

$$SubTriples((o\ type\ c)) = \{(o\ type\ c') \mid c' \leq^*_{cl} c\}$$
$$SubTriples((o\ pr\ o')) = \{(o\ pr'\ o') \mid pr' \leq^*_{pr} pr\}$$

If A is a set of instance triples, we define: $Sub\ Triples(A) = \bigcup_{t \in A} Sub\ Triples(t)$. Given two triples t and t', we shall write: $t \leq t'$ iff $t \in SubTriples(t')$.

Now, we introduce the notion of TFP-partition (from *T*rue *F*alse *P*ossible) which is fundamental for our work. The main idea is to partition the set of cartesian instance triples B_K into three pairwise disjoint subsets: *True*, *False*, and *Possible* instance triples:

- the first comprises I_K and the inferred instance triples (i.e. $\mathcal{C}_i(K)$),
- the second comprises instance triples which are not true (denoted by F_K), and
- the last comprises instance triples (denoted by P_K) that are possible.

Definition 5 (TFP-partition) An *TFP-partition of* B_K is a three-fold partition of B_K, denoted by $(\mathcal{C}_i(K),\ F_K,\ P_K)$, that satisfies the following:

(i) $\mathcal{C}_i(K) = \mathcal{C}(K) \cap B_K$.
(ii) F_K is a *lower set*[3] wrt $\leq$, i.e. $SubTriples(F_K) = F_K$.
(iii) If an element of B_k is not valid then it belongs to F_K, i.e. $Invalid(K) \subseteq F_K$.

Note that $\mathcal{C}_i(K)$ is an *upper set*[4] wrt $\leq$, as consequence of the derivation rules (i) and (ii) of RDF/S semantics (see Section 2) and the fact that $\mathcal{C}(K)$ is closed with respect to the closure operator $\mathcal{C}$. Definition 5 (ii) actually says that: (a) if $(o\ type\ c_1) \in F_K$ and $c_2 \leq^*_{cl} c_1$ then it should hold $(o\ type\ c_2) \in F_K$, and (b) if $(o\ pr_1\ o') \in F_K$ and $pr_2 \leq^*_{pr} pr_1$ then it should be $(o\ pr_2\ o') \in F_K$. It is not hard to see that it would be "irrational" if these two were not satisfied.

[3] A *lower set* (else called *downward closed set*) is a subset Y of a given partially ordered set $(X, \leq)$ s.t. for all elements x and y, if $x \leq y$ and y is an element of Y, then x is also in Y.

[4] *Upper set* is the dual notion of lower set.

It follows from Definition 5(ii) and the fact that a TFP-partition is a partition that it does not exist $m \in F_K$ and $p \in P_K$ such that $p \leq m$.

Clearly, the triples in P_K fall into the following two categories:

1. class instance triples in $((Inst_K \cap URI) \times \{type\} \times C_K)$,
2. *valid to add* property instance triples, i.e. property instance triples whose addition in K would not add any inferred class instance triple to $\mathcal{C}_i(K)$.

Figure 6 illustrates some the above notions and notations using Venn Diagrams. Two lemmas follow.

Lemma 1 ($P_K \cup \mathcal{C}_i(K)$ is an Upper Set wrt $\leq$)

1. *If* $(o\ type\ c_2) \in P_K$ *and* $c_2 \leq^*_{cl} c_1$ *then* $(o\ type\ c_1) \in (P_K \cup \mathcal{C}_i(K))$.
2. *If* $(o\ pr_2\ o') \in P_K$ *and* $pr_2 \leq^*_{pr} pr_1$ *then* $(o\ pr_1\ o') \in (P_K \cup \mathcal{C}_i(K))$.

Lemma 2 (P_K is interval-based wrt $\leq$)

1. *If* $c_1 \leq^*_{cl} c_2 \leq^*_{cl} c_3$ *and* $(o\ type\ c_1), (o\ type\ c_3) \in P_K$ *then* $(o\ type\ c_2) \in P_K$.
2. *If* $pr_1 \leq^*_{pr} pr_2 \leq^*_{pr} pr_3$ *and* $(o\ pr_1\ o'), (o\ pr_3\ o') \in P_K$ *then* $(o\ pr_2\ o') \in P_K$.

Lemma 2 states that if t and t' belong to P_K then all t_x in the interval $[t, t']$ (defined over the $\leq$ partial order) belong to P_K, too. This lemma can be exploited for specifying compact (interval-based) representations of P_K (more at Sect. 10.2).

3.1 Capturing Application Hypotheses/Scenarios with TFP-partitions

[MSA case].
Consider a KB K that satisfies the *Maximum Specificity Assumption* (*MSA*), i.e. all instances have been described with the most specific classes or properties of the schema that hold in the application domain. We can capture this case with a TFP-partition in which $P_K = \emptyset$. Since a TFP-partition is a partition, and $P_K = \emptyset$, it follows that all elements of B_K that are not in $\mathcal{C}_i(K)$, should be considered as false, i.e. $F_K = B_K \setminus \mathcal{C}_i(K)$, and thus the TFP-partition of B_K is:

$$(\mathcal{C}_i(K), F_K, P_K) = (\mathcal{C}_i(K), B_K \setminus \mathcal{C}_i(K), \emptyset)$$

It is not hard to verify that the above is a TFP-partition. Indeed (ii) and (iii) of Definition 5 are satisfied. Regarding (ii), it holds that $SubTriples(B_K \setminus \mathcal{C}_i(K)) = B_K \setminus \mathcal{C}_i(K)$, because if $t \in B_K \setminus \mathcal{C}_i(K)$ and $t' \leq t$, then $t' \in B_K \setminus \mathcal{C}_i(K)$, due to derivation rules (iii) and (iv) of RDF/S semantics (see Section 2). Regarding (iii), note that $\mathcal{C}_i(K) \cap Invalid(K) = \emptyset$,

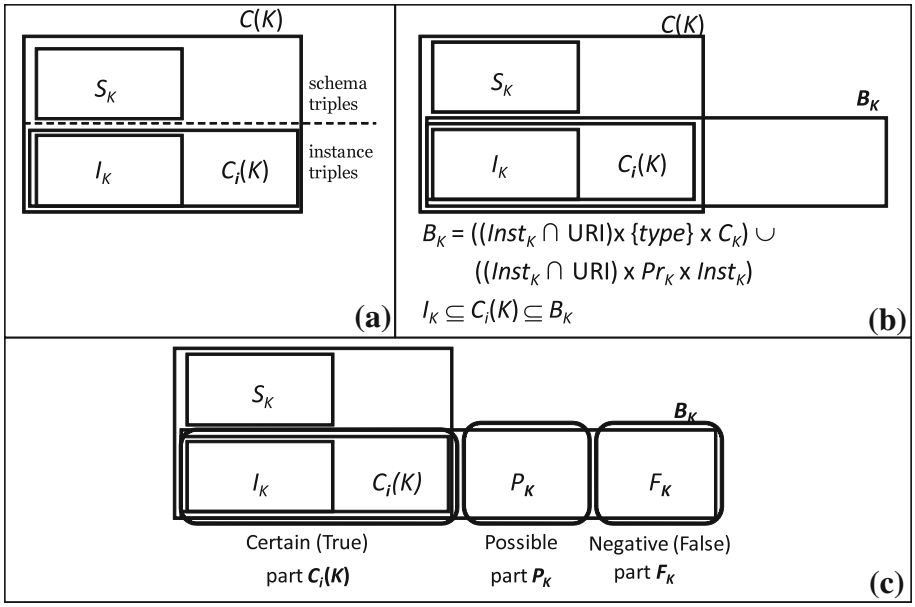

Fig. 6 TFP-partition illustrated with Venn Diagrams

due to derivation rules (v) and (vi) of RDF/S semantics, thus, $Invalid(K) \subseteq B_K \setminus \mathcal{C}_i(K)$.

[Open World case].
Now consider the other extreme case where the *MSA* does not hold for any instance. For example, consider that we start from a KB for which we know nothing regarding its completeness or specificity, and we want to consider that every valid instance triple that can be formed using the ontology and is not certain, it is possible. We can capture this knowledge by a TFP-partition whose F_K contains only the invalid instance triples of B_K, i.e. $F_K = Invalid(K)$. Since a TFP-partition is a partition of B_K, it follows that $P_K = B_K \setminus (\mathcal{C}_i(K) \cup F_K) = B_K \setminus (\mathcal{C}_i(K) \cup Invalid(K))$, meaning that P_K includes all valid instance triples of B_K that do not belong to $\mathcal{C}_i(K)$, and hence the TFP-partition of B_K is:

$$(\mathcal{C}_i(K), F_K, P_K) = (\mathcal{C}_i(K), Invalid(K), B_K \setminus (\mathcal{C}_i(K) \cup Invalid(K)))$$

It is not hard to verify that the above is a TFP-partition[5].

[Mixed case].
Apart from the above (extreme) cases, TFP-partitionscan be used to capture various application-specific assumptions, e.g. cases where we know that *MSA* holds only for a *part* of the KB.

[5] Definition 5(ii) is satisfied because if it holds that $\neg valid(o, pr, o', K)$ and $pr' \leq^*_{pr} pr$ then certainly it holds that $\neg valid(o, pr', o', K)$, due to (ii) of Definition 2. Thus, $SubTriples(Invalid(K)) = Invalid(K)$. Definition 5(iii) is obvious.

4 Transitioning from one TFP-partition to Another

Here we will formalize schema migrations, by conceiving them as *transitions* between TFP-partitionswhich should be governed by a few fundamental postulates. For reasons of readability, we will first focus on migrations to *backwards compatible* schemas.

Definition 6 (*Backwards compatibility*) Let S and S' be two sets of schema triples. S' is *backwards compatible* with S, denoted by $S \sqsubseteq S'$, iff $\mathcal{C}(S) \subseteq \mathcal{C}(S')$.

For example, for the KBs of Fig. 5 we have $S_K \sqsubseteq S_{K'}$. Note that we can check if a set of schema triples S' is backwards compatible with another set of schema triples S by computing the Delta function $\Delta_d(S \rightarrow S')$, defined as follows [32,33]:

$$\Delta_d(S \rightarrow S') = \{Add(t) \mid t \in S' \setminus \mathcal{C}(S)\} \cup \{Del(t) \mid t \in S \setminus \mathcal{C}(S')\}$$

Note that $Add(t)$ and $Del(t)$ are not functions on t but strings, where t is replaced by the appropriate schema triple.

Proposition 1 *$S \sqsubseteq S'$ iff $\Delta_d(S \rightarrow S')$ contains only Add operations.*

Consider that we want to migrate the instance triples of a KB $K = (S_K, I_K)$, to a schema $S_{K'} \sqsupseteq S_K$, reaching to a KB $K' = (S_{K'}, I_K)$. It is not hard to see that it holds $B_{K'} \supseteq B_K$. Since $S_{K'}$ is backwards compatible with S_K, every b that belongs to B_K certainly belongs to $B_{K'}$. The superset relationship between B_K and $B_{K'}$ can be strict (i.e. $B_{K'} \supset B_K$) because $S_{K'}$ can contain new elements (classes or properties) that could be used for generating instance triples to be added to $B_{K'}$.

Proposition 2 *If* $S_K \sqsubseteq S_{K'}$ *then* $B_{K'} \supseteq B_K$.

Suppose that we know the TFP-partition of K, i.e. $(\mathcal{C}_i(K), F_K, P_K)$. Our objective is to define the new TFP-partition, i.e. we want to define the transition:

$$(\mathcal{C}_i(K), F_K, P_K) \rightsquigarrow (\mathcal{C}_i(K'), F_{K'}, P_{K'})$$

By migrating I_K to $S_{K'}$, we can get $\mathcal{C}(K')$, and consequently $\mathcal{C}_i(K')$. The rising question is how $F_{K'}$ and $P_{K'}$ are defined.

As we mentioned in the introductory section, we can have conflicts among:

(a) new positive knowledge inferable from the instance triples and the new schema,
(b) new "negative" information inferable from the past negative instance triples and the new schema, and
(c) the previously computed possibilities (possible refinements).

We resolve such conflicts by considering that:
(a) has higher priority than (b), and
(b) has higher priority than (c).

These priorities can be formally expressed by two postulates that concern the transition from one TFP-partition to another.

Definition 7 (*TFP-partition Evolution Postulates*) A transition $(\mathcal{C}_i(K), F_K, P_K) \rightsquigarrow (\mathcal{C}_i(K'), F_{K'}, P_{K'})$ is *consistent* if the following postulates are satisfied:

(Π1) $\mathcal{C}_i(K')$ does not depend on F_K or P_K.
Priority of the positive knowledge inferrable from the instance triples and the new schema.
(Π2) $F_K \cap P_{K'} = \emptyset$.
Past negative information cannot become possible.

Postulate Π1 gives priority to the new positive knowledge over past negative or possible knowledge. It is consistent with (and reminiscent of) the principle "Recent knowledge prevails the old one" (also, called "Principle of Success" [1] and "Primacy of New Information" [6]).

Postulate Π2 says that past negative information cannot become possible. This implies, as stated and proved by the Proposition 3 that follows, that past negative information is *preserved* as long as it does not contradict with the new positive knowledge.

Proposition 3 (Inertia rule for negatives) *In the context of a consistent transition* $(\mathcal{C}_i(K), F_K, P_K) \rightsquigarrow (\mathcal{C}_i(K'), F_{K'}, P_{K'})$, *it follows that:* $F_K \cap P_{K'} = \emptyset$ *(i.e.* Π2*) iff* $(F_K \backslash \mathcal{C}_i(K')) \subseteq F_{K'}$.

Below, we present also an example justifying Postulate Π2.

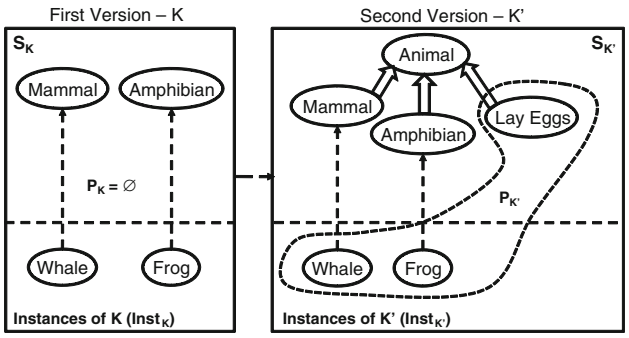

Fig. 7 Motivating example for Postulate Π2

Example 2 Consider Fig. 7. It holds that $I_K = I_{K'} = \{$(`Whale type Mammal`), (`Frog type Amphibian`)$\}$. Assuming that the *MSA* holds for KB K, it follows that $F_K = \{$(`Whale type Amphibian`), (`Frog type Mammal`)$\}$. In $S_{K'}$, two new classes are added `Animal` and `Lay Eggs`, as well as the relationships `Mammal` $\leq_{cl}$ `Animal`, `Amphibian` $\leq_{cl}$ `Animal`, and `Lay Eggs` $\leq_{cl}$ `Animal`. Obviously, we want $F_{K'} = F_K$ and $P_{K'} = \{$(`Whale type Lay Eggs`), (`Frog type Lay Eggs`)$\}$. Note that it holds $F_K \cap P_{K'} = \emptyset$.

Based on the above postulates, we will specify how a transition can be performed. At first we introduce some notational conventions.

Notational conventions: In the context of two KBs K and K', and unless otherwise indicated, with c we will denote a class in both C_K and $C_{K'}$, while with c' we will denote a class in $C_{K'}$ (that could however belong also to C_K). Analogously, with pr we will denote a property in both Pr_K and $Pr_{K'}$, while with pr' we will denote a property in $Pr_{K'}$ (that could belong also to Pr_K). In addition, with $\leq^*_{cl}$ and $\leq^*_{pr}$, we will refer to the relations of the new schema $S_{K'}$. Further, we consider that the $\leq^*_{cl}$ and $\leq^*_{pr}$ relations, used in defining *SubTriples*(A), refer to the relations of the new schema $S_{K'}$.

Definition 8 (*Derivation of negatives at a transition*) Consider a TFP-partition $(\mathcal{C}_i(K), F_K, P_K)$ based on a schema S_K and suppose that we want to define the TFP-partition after migrating I_K to a backwards compatible schema $S_{K'}$. Consider deriving $F_{K'}$ using the following rules:

(R1) If $(o\ type\ c) \in F_K$, $c' \leq^*_{cl} c$, and $(o\ type\ c) \notin \mathcal{C}_i(K')$ then $(o\ type\ c') \in F_{K'}$.
(R2) If $(o\ pr\ o') \in F_K$, $pr' \leq^*_{pr} pr$, and $(o\ pr\ o') \notin \mathcal{C}_i(K')$ then $(o\ pr'\ o') \in F_{K'}$.
(R3) If $(o\ pr'\ o') \in B_{K'}$ and $\neg valid(o, pr', o', K')$ then $(o\ pr'\ o') \in F_{K'}$.

Essentially, the above rules produce the following set of instance triples:

$$F_{K'} = Invalid(K') \cup SubTriples(F_K \setminus \mathcal{C}_i(K'))$$

It is not hard to see that the $F_{K'}$ derived by the above rules satisfies the following: $F_{K'}$ and $\mathcal{C}_i(K')$ are disjoint, $F_{K'}$ is a lower set, and $F_{K'}$ contains all invalid property instance triples of $B_{K'}$.

Proposition 4 (Derivation of negatives at a transition) *The negative part of the target TFP-partition of a consistent transition, always satisfies the rules of Definition 8.*

Having derived $\mathcal{C}_i(K')$ and $F_{K'}$, we can now derive $P_{K'}$, and thus obtain the TFP-partition of $B_{K'}$, by excluding from $B_{K'}$ all triples in $\mathcal{C}_i(K')$ or in $F_{K'}$.

Definition 9 (*Deriving $P_{K'}$ through $B_{K'}, C_i(K')$ and $F_{K'}$*) If we know $C_i(K')$ and $F_{K'}$, then we can derive $P_{K'}$ as follows: $P_{K'} = B_{K'} \setminus (F_{K'} \cup \mathcal{C}_i(K'))$.

We can now state and prove the following important proposition.

Proposition 5 (Transition correctness) *The derivation of $F_{K'}$ using the rules of Definition 8, and the derivation of $P_{K'}$ by Definition 9, yields a three-fold partition that is a TFP-partition and $(\mathcal{C}_i(K), F_K, P_K) \rightsquigarrow (\mathcal{C}_i(K'), F_{K'}, P_{K'})$ is a consistent transition, i.e. it respects postulates* $\Pi 1$ *and* $\Pi 2$. *We will denote this particular transition by:*

$$(\mathcal{C}_i(K), F_K, P_K) \Rrightarrow (\mathcal{C}_i(K'), F_{K'}, P_{K'})$$

The derived TFP-partition, not only makes the transition consistent, but it has the *minimum set of negatives* among all TFP-partitionswhich constitute consistent transitions (it follows directly from Proposition 4).

Algorithmically, from F_K, K, and $S_{K'}$, we can produce $F_{K'}$ by straightforwardly applying the rules of Definition 8.

5 On Managing TFP-partitions without F-sets

Suppose that we want to migrate the instance triples of a KB $K = (S_K, I_K)$, to a schema $S_{K'}$ such that $S_K \sqsubseteq S_{K'}$. To compute $P_{K'}$, one approach is to apply the method described in the previous section. However, the shortcoming of that approach is that it requires computing the whole TFP-partition and to keep F_K stored, something that could be very space consuming. Therefore, below we investigate whether we can achieve our goal (derive $P_{K'}$) *without having to compute* $F_{K'}$, and *without having kept* F_K.

We prefer an approach requiring the availability only of P_K, because it is reasonable to assume that $|P_K| < |F_K|$, meaning that a P_K-based approach for computing $P_{K'}$ requires fewer space than a F_K-based approach (like the one described in the previous section). After all, the ideal state of a curated KB is a state that satisfies the *MSA*, and such a state requires keeping only K (since P_K is empty at that case). Instead, a F_K-based approach would require keeping a non-empty (and possibly very large) F_K even if the *MSA* holds. In general, the more we approach the *MSA*, the fewer storage is required for a P_K-based approach, while the opposite holds for the F_K-based approach. Also note that the lifecycle management process that we propose at Sect. 9 aims at reducing possibilities and approaching a state satisfying *MSA*.

Thus, we introduce what we call *extended KB*.

Definition 10 (*Extended KB*) An *extended KB*, for short *eKB*, is a pair $\mathcal{E} = (K, P_K)$, where K is a KB and P_K is a set of possible instance triples s.t. $P_K \cap \mathcal{C}(K) = \emptyset$.

In fact, the instance triples of K, and the set P_K, are two parts of the TFP-partition of K. Below, we will try to compute $P_{K'}$ just from K, P_K, and $S_{K'}$. The envisioned process will be *correct* if it gives the same result ($P_{K'}$) as Definition 9.

In general, $P_{K'}$ can be produced by adding and deleting triples to/from P_K, i.e. we can write

$$P_{K'} = (P_K \setminus P_{K\_Del}) \cup P_{K\_Add}$$

where P_{K_Del} are the elements of P_K that should be deleted from it and P_{K_Add} are the elements that should be added to P_K. Due to priority of new knowledge, we certainly know that

$$P_{K\_Del} \supseteq P_K \cap \mathcal{C}_i(K')$$

In fact,

$$P_{K\_Del} = (P_K \cap \mathcal{C}_i(K')) \cup (P_K \cap F_{K'})$$

This is because, the only triples that can be deleted from P_K belong to $\mathcal{C}_i(K')$ or $F_{K'}$. Regarding P_{K_Add}, it should contain instance triples that involve the new classes and properties. The following two Propositions indicate the instance triples to be added to P_{K_Add}.

Proposition 6 *Suppose we are in the context of $(\mathcal{C}_i(K), F_K, P_K) \Rrightarrow (\mathcal{C}_i(K'), F_{K'}, P_{K'})$. For a new class $c' \in C_{K'} \setminus C_K$, it holds that: $(o\ type\ c') \in P_{K'}$ iff:*

(i) $o \in Inst_K \cap URI$, *and*
(ii) *for all $c \in C_K$ s.t. $c' \leq^*_{cl} c$, it holds that $(o\ type\ c) \in (\mathcal{C}_i(K') \cup P_K)$, and*
(iii) $(o\ type\ c') \notin \mathcal{C}_i(K')$.

Proposition 7 *Suppose we are in the context of $(\mathcal{C}_i(K), F_K, P_K) \Rrightarrow (\mathcal{C}_i(K'), F_{K'}, P_{K'})$. For a new property $pr' \in Pr_{K'} \setminus Pr_K$, it holds that: $(o\ pr'\ o') \in P_{K'}$ iff:*

(i) $o \in URI$ *and* $valid(o,\ pr',\ o',\ K')$, *and*
(ii) *for all* $pr \in Pr_K$ *s.t.* $pr' \leq^*_{pr} pr$, *it holds that* $(o\ pr\ o') \in (\mathcal{C}_i(K') \cup P_K)$, *and*
(iii) $(o\ pr'\ o') \notin \mathcal{C}_i(K')$.

5.1 Algorithmic Perspective

Below we present Algorithm 1, called *Produce_Possibilities*, which takes as input a KB K, its set of possible instance triples P_K, and a new set of schema triples $S_{K'}$ (s.t. $S_K \sqsubseteq S_{K'}$), where $K' = (S_{K'}, I_K)$ is a (valid) KB, and it produces the set of possible instance triples $P_{K'}$ for the new KB K'.

As the following Proposition states, Algorithm *Produce_Possibilities*$(K, P_K, S_{K'})$ is *correct*, i.e. that it produces the same $P_{K'}$ as that defined in Definition 9 without the need of computing $B_{K'}$ and $F_{K'}$.

Proposition 8 *If* $(\mathcal{C}_i(K), F_K, P_K) \Rrightarrow (\mathcal{C}_i(K'), F_{K'}, P_{K'})$, *then* $P_{K'} = Produce\_Possibilities(K,\ P_K,\ S_{K'})$.

Part A (lines 5–6) of Algorithm 1 follows from Proposition 6 and concerns new classes $c' \in NC$. If a new class c' has superclasses c (which are existing classes in K), we have to check if P_K or $\mathcal{C}_i(K')$ includes triples of the form $(o\ type\ c)$ for each superclass c, where $o \in Inst_K \cap URI$. Only if this is true and $(o\ type\ c')$ does not belong to $\mathcal{C}_i(K')$, we can safely add triples of the form $(o\ type\ c')$ to P_{K_Add}. If a new class c' has no superclasses that are existing classes in K, we just have to check whether a triple of the form $(o\ type\ c')$ does not belong to $\mathcal{C}_i(K')$ before adding it to P_{K_Add}.

Part B (lines 7–8), concerns existing classes $c_1 \in C_K$, for which we have to add to P_{K_Del} all those triples $(o\ type\ c_1) \in P_K$, if there is a class $c_2 \in C_K$ such that $c_1 \leq^*_{cl} c_2$ in K' and $(o\ type\ c_2) \notin (\mathcal{C}_i(K') \cup P_K)$. This is because, since $S_K \sqsubseteq S_{K'}$, it holds that $(o\ type\ c_2) \notin \mathcal{C}_i(K)$. Thus, $(o\ type\ c_2) \in F_K$. Therefore, it follows from Rule $R1$ of Proposition 4 at $(o\ type\ c_1) \in F_{K'}$.

Part C (lines 10–11) follows from Proposition 7 and concerns new properties $pr' \in NP$. If a new property pr' has superproperties pr (which are existing properties in K), we have to check if P_K or $\mathcal{C}_i(K')$ includes triples of the form $(o\ pr\ o')$ for each superproperty pr. Only if this is true, it holds that $valid(o,\ pr',\ o',\ K')$, and $(o\ pr'\ o')$ does not belong to $\mathcal{C}_i(K')$, we can safely add triples of the form $(o\ pr'\ o')$ to P_{K_Add}. If a new property has no superproperties that are existing properties in K, we just have to check whether it holds $valid(o,\ pr',\ o',\ K')$ and $(o\ pr'\ o')$ does not belong to $\mathcal{C}_i(K')$ before adding $(o\ pr'\ o')$ to P_{K_Add}.

Part D (lines 12–13) concerns existing properties $pr_1 \in Pr_K$, for which we have to add to P_{K_Del} all those triples $(o\ pr_1\ o') \in P_K$, if there is a property $pr_2 \in Pr_K$ such that $pr_1 \leq^*_{pr} pr_2$ in K' and $(o\ pr_2\ o') \notin (\mathcal{C}_i(K') \cup P_K)$. This is because, since $S_K \sqsubseteq S_{K'}$, it holds that $(o\ pr_2\ o') \notin \mathcal{C}_i(K)$. Thus, $(o\ pr_2\ o') \in F_K$. Therefore, it follows from Rule $R2$ of Proposition 4 that $(o\ pr_1\ o') \in F_{K'}$.

In line 14 of Algorithm 1, we add to P_{K_Del} the set $P_K \cap \mathcal{C}_i(K')$, because all instance triples that belong to $\mathcal{C}_i(K')$ have to be removed from P_K. At the end (lines 15–16), we have to update P_K by adding to it the P_{K_Add} set and by removing from it the P_{K_Del} set. Then, we return $P_{K'}$. The execution order of the above parts A, B, C, and D does not matter since in any order the output result $P_{K'}$ is the same.

Examples of applying the algorithm are given in Appendix B.

Proposition 9 *The time complexity of Algorithm 1 is* $O(|Inst_K|^2 * M^2 * (|K'|^2 + |P_K|)$, *where* $M = max(|C_{K'}|, |Pr_{K'}|)$.

6 Non-backwards Compatible Schema Evolution

Here we consider the case where the next schema version $S_{K'}$ of a KB K', is not backwards compatible with S_K, i.e. $S_K \not\sqsubseteq S_{K'}$ (and consequently $\mathcal{C}(S_K) \not\subseteq \mathcal{C}(S_{K'})$), however the instance triples in I_K are transferred to K', i.e. $K' = (S_{K'}, I_K)$, and K' is a (valid) KB.[6] The changes between $S_{K'}$ and S_K may include deletions of classes, deletions of properties, changes in the subClassOf/subPropertyOf relations, or changes in the domain and range of properties. The elements of I_K refer to (usually leaf) classes and properties of S_K which are preserved in K' automatically by the semantics of RDF/S[7] [13].

Let $K' = (S_{K'}, I_K)$ be a KB. Obviously, it may hold that $B_K \not\subseteq B_{K'}$. For example, if $(o\ type\ c) \in B_K$ and $c \in C_K \setminus C_{K'}$ then $(o\ type\ c) \notin B_{K'}$.

Our goal is to describe how the rules that are used to derive $F_{K'}$ are modified. In brief, the following remain the same:
Definition 7 defining the postulates $\Pi 1$ and $\Pi 2$,
Definition 8 and Proposition 4 regarding the rules $R1$, $R2$, and $R3$, and
Definition 9 defining $P_{K'}$.
Proposition 3 essentially remains the same, but we have to replace $F_K \setminus \mathcal{C}_i(K') \subseteq F_{K'}$ by $(F_K \setminus \mathcal{C}_i(K')) \cap B_{K'} \subseteq F_{K'}$. This is because there may exist an instance triple $t \in F_K$ but $t \notin B_{K'}$, due to deletions of classes and properties in $S_{K'}$.

However, and regarding the transition postulates, we have to add a *new postulate* $\Pi 3$ applicable only to non-backwards compatible schema evolution case. Postulate $\Pi 3$ expresses that if a triple $t \in B_{K'}$ that existed in $\mathcal{C}_i(K)$, does not exist in $\mathcal{C}_i(K')$ then t should go to $F_{K'}$.

[6] We discuss the notion of *validity* at Sect. 11 along with the discussion of the related works that focus on that issue.

[7] If $(o\ type\ c) \in K'$ then $(c\ type\ Class) \in \mathcal{C}(K')$ and if $(o\ pr\ o') \in K'$ then $(pr\ type\ Property) \in \mathcal{C}(K')$.

Algorithm 1 *Produce_Possibilities*$(K, P_K, S_{K'})$

Input: a KB K, its set of possible instance triples P_K, and a set of schema triples $S_{K'}$ s.t. $S_K \sqsubseteq S_{K'}$ and $K' = (S_{K'}, I_K)$ is a (valid) KB

Output: the set of possible instance triples $P_{K'}$ of the KB K'

(1) $K' = (S_{K'}, I_K)$;

(2) $P_{K\_Add} = \emptyset$;

(3) $P_{K\_Del} = \emptyset$;

/* FOR CLASS INSTANCES: */

(4) $NC = C_{K'} \setminus C_K$; /* new classes appearing in K' */

/* PART A: New classes */

(5) For all $(c' \in NC)$ do /* for each new class */

(6) $P_{K\_Add} = P_{K\_Add} \cup \{(o\ type\ c') \mid o \in Inst_K \cap URI,$
$\forall\ c \in C_K$ s.t. $c' \leq^*_{cl} c$ it holds that $(o\ type\ c) \in (\mathcal{C}_i(K') \cup P_K)$, and
$(o\ type\ c') \notin \mathcal{C}_i(K')\}$;

/* PART B: Existing classes */

(7) For all $(c_1 \in C_K)$ do

/* Moving class instance triples from P_K to $F_{K'}$ due to Rule $R1$. */

(8) $P_{K\_Del} = P_{K\_Del} \cup \{(o\ type\ c_1) \in P_K \mid c_2 \in C_K,\ c_1 \leq^*_{cl} c_2$, and
$(o\ type\ c_2) \notin (P_K \cup \mathcal{C}_i(K'))\}$;

/* FOR PROPERTIES: */

(9) $NP = Pr_{K'} \setminus Pr_K$; /* new properties appearing in K' */

/* PART C: New properties */

(10) For all $(pr' \in NP)$ do /* for each new property */

(11) $P_{K\_Add} = P_{K\_Add} \cup \{(o\ pr'\ o') \mid o \in URI,\ valid(o,\ pr',\ o', K'),$
$\forall\ pr \in Pr_K$ s.t. $pr' \leq^*_{pr} pr$ it holds that $(o\ pr\ o') \in (\mathcal{C}_i(K') \cup P_K)$, and
$((o\ pr'\ o') \notin \mathcal{C}_i(K')\}$;

/* PART D: Existing properties */

(12) For all $(pr_1 \in Pr_K)$ do

/* Moving property instance triples from P_K to $F_{K'}$ due to Rule $R2$. */

(13) $P_{K\_Del} = P_{K\_Del} \cup \{(o\ pr_1\ o') \in P_K \mid pr_2 \in Pr_K, pr_1 \leq^*_{pr} pr_2$, and
$(o\ pr_2\ o') \notin (P_K \cup \mathcal{C}_i(K'))\}$;

(14) $P_{K\_Del} = P_{K\_Del} \cup (P_K \cap \mathcal{C}_i(K'))$;

(15) $P_{K'} = P_K \setminus P_{K\_Del}$;

(16) $P_{K'} = P_{K'} \cup P_{K\_Add}$;

(17) Return $P_{K'}$;

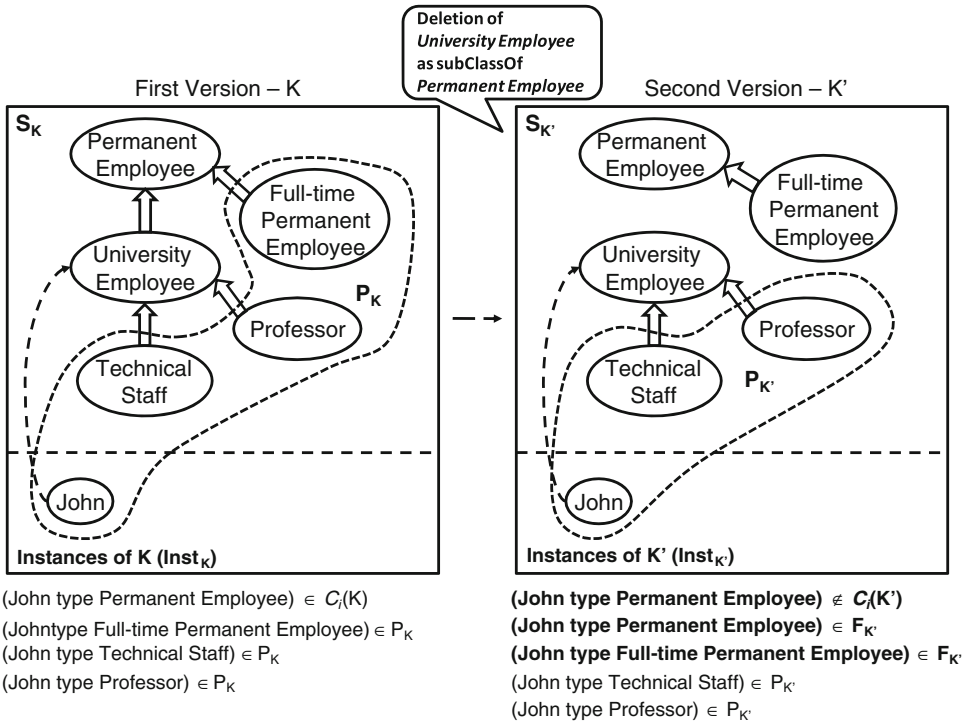

Fig. 8 Certain modification inheritance rules

Definition 11 (*NBC Evolution Postulates*)[8] A transition $(\mathcal{C}_i(K), F_K, P_K) \rightsquigarrow (\mathcal{C}_i(K'), F_{K'}, P_{K'})$ is *consistent* if, apart from the postulates $\Pi 1$ and $\Pi 2$ of Definition 7, the following postulate is satisfied.

($\Pi 3$) All elements $t \in B_{K'}$ s.t. $t \in \mathcal{C}_i(K) \setminus \mathcal{C}_i(K')$ are placed to $F_{K'}$.

Note that postulate $\Pi 3$ is not needed in the backwards compatible schema evolution case, because in this case it holds that $\mathcal{C}_i(K) \subseteq \mathcal{C}_i(K')$.

Example 3 Consider the scenario shown in Fig. 8, where $S_K \not\sqsubseteq S_{K'}$. Suppose that {(John type Full-time Permanent Employee), (John type Technical Staff), (John type Professor)} $\in P_K$. Note that the specialization relationship University Employee $\leq_{cl}$ Permanent Employee, which exists in S_K, has been deleted in $S_{K'}$. Note that (John type Permanent Employee) $\in \mathcal{C}_i(K)$, while (John type Permanent Employee) $\notin \mathcal{C}_i(K')$. According to postulate $\Pi 3$, (John type Permanent Employee) should go to $F_{K'}$.

The following proposition provides an equivalent form of postulate $\Pi 3$.

Proposition 10 *In the context of a transition* $(\mathcal{C}_i(K), F_K, P_K) \rightsquigarrow (\mathcal{C}_i(K'), F_{K'}, P_{K'})$, *it follows that:* $(\mathcal{C}_i(K) \setminus \mathcal{C}_i(K')) \cap B_{K'} \subseteq F_{K'}$ ($\Pi 3$) *iff* $\mathcal{C}_i(K) \cap P_{K'} = \emptyset$.

Based on $\Pi 3$, we provide the following rules that produce (additionally to rules $R1$, $R2$, and $R3$) elements of $F_{K'}$, for the classes and properties that exist in K'.

Proposition 11 (Certain modification inheritance rules) *The negative part of the target TFP-partition of a consistent transition, always satisfies the following rules:*[9]

(R4) If $(o\ type\ c) \in \mathcal{C}_i(K)$, $c' \leq^*_{cl} c$, *and* $(o\ type\ c) \notin \mathcal{C}_i(K')$ *then* $(o\ type\ c') \in F_{K'}$.
(R5) If $(o\ pr\ o') \in \mathcal{C}_i(K)$, $pr' \leq^*_{pr} pr$, *and* $(o\ pr\ o') \notin \mathcal{C}_i(K')$ *then* $(o\ pr'\ o') \in F_{K'}$.

Essentially, rules $R1 - R5$ produce the following set of instance triples:

$$F_{K'} = Invalid(K') \cup SubTriples((F_K \setminus \mathcal{C}_i(K')) \cap B_{K'}) \cup SubTriples((\mathcal{C}_i(K) \setminus \mathcal{C}_i(K')) \cap B_{K'})$$

Obviously, a $F_{K'}$ derived by the rules $R1$-$R5$ respects the following validity constraints of TFP-partition (Definition 5): $F_{K'}$ and $\mathcal{C}_i(K')$ are disjoint, $F_{K'}$ is a lower set, and $F_{K'}$ contains all invalid property instance triples of $B_{K'}$.

It is easy to see that the derivation of $F_{K'}$ by the rules $R1$, $R2$, and $R3$ of Proposition 4 and $R4$, $R5$ of Proposition 11, as well as the derivation of $P_{K'}$ by Definition 9, yield a TFP-partition that makes the transition $(\mathcal{C}_i(K), F_K, P_K) \rightsquigarrow (\mathcal{C}_i(K'), F_{K'}, P_{K'})$ *consistent*, i.e. it respects postulates $\Pi 1$

[8] NBC stands for non-backwards compatible.

[9] The relationships $\leq^*_{cl}$ and $\leq^*_{pr}$ hold in K'.

and Π2 of Definition 7 and Π3 of Definition 11. Moreover, and analogously to the backward compatible case, the derived TFP-partition has the *minimum set of negatives* (again we will use $\Rrightarrow$ to denote this particular transition).

Example 4 Continuing Example 3, according to Rule $R4$, (`John type Permanent Employee`) should go to $F_{K'}$. Further, according to Rule $R4$, (`John type Full time Permanent Employee`) must be moved from P_K to $F_{K'}$.

Now, similarly to the case of backwards compatible schema evolution, we will try to compute $P_{K'}$ from K, P_K, and $S_{K'}$ without having to know or compute neither F_K nor $F_{K'}$. The envisioned process will be *correct* if it gives the same result ($P_{K'}$) as Definition 9.

In general, $P_{K'}$ can be produced by adding and deleting triples to/from P_K, i.e. we can write $P_{K'} = (P_K \setminus P_{K\_Del}) \cup P_{K\_Add}$, where P_{K_Del} are the elements of P_K that should be deleted from it and P_{K_Add} are the elements that should be added to P_K. In fact, $P_{K\_Del} = (P_K \cap \mathcal{C}(K')) \cup (P_K \cap F_{K'}) \cup \{t \in P_K \mid t \notin B_{K'}\}$[10]. This is because, the only triples that can be deleted from P_K belong to $\mathcal{C}(K') \cup F_{K'}$ or do not belong to $B_{K'}$. Regarding P_{K_Add}, it should contain instance triples that involve the new classes and properties. The following two Propositions indicate the instance triples to be added to P_{K_Add}.

Proposition 12 *The same as Proposition 6 but now applies to KBs $K = (S_K, I_K)$ and $K' = (S_{K'}, I_K)$, where S_K and $S_{K'}$ are not necessarily backwards compatible.*

Proposition 13 *The same as Proposition 7 but now applies to KBs $K = (S_K, I_K)$ and $K' = (S_{K'}, I_K)$, where S_K and $S_{K'}$ are not necessarily backwards compatible.*

Below we present Algorithm 2, called *Produce_Possibilities$_{NBC}$*, which takes as input a KB K, its set of possible instance triples P_K, and a new set of schema triples $S_{K'}$ s.t. $K' = (S_{K'}, I_K)$ is a (valid) KB, and produces the set of possible instance triples $P_{K'}$ for the new KB K'.

The following Proposition shows that Algorithm *Produce_Possibilities$_{NBC}$*$(K, P_K, S_{K'})$ is *correct*.

Proposition 14 *Let $K = (S_K, I_K)$ and $K' = (S_{K'}, I_K)$ be two valid KBs.*
If $(\mathcal{C}_i(K), F_K, P_K) \Rrightarrow (\mathcal{C}_i(K'), F_{K'}, P_{K'})$, then
$P_{K'} = Produce\_Possibilities_{NBC}(K, P_K, S_{K'})$.

As we can see, the only differences between Algorithm 2 and Algorithm 1 are the new parts C and F (see bold lines).

Note that lines (5–8) of Algorithm 2 and lines (5–8) of Algorithm 1, as well as lines (12–15) of Algorithm 2 and lines (10–13) of Algorithm 1 are the same. However, their proof is different. This is because the second ontology version is not necessarily backwards compatible with the first one and the instance triples $(o\ type\ c_2)$ or $(o\ pr_2\ o')$ may belong to $\mathcal{C}_i(K)$ and not to $\mathcal{C}_i(K')$.

Part C (lines 9–10) concerns class instance triples of the form $(o\ type\ c)$ that belong to P_K and refer to classes c that have been deleted in K'. These class instance triples have to be added to P_{K_Del} set, as they do not belong to $B_{K'}$.

Part F (lines 16–17) concerns property instance triples of the form $(o\ pr\ o')$ that belong to P_K and refer to properties pr that either have been deleted in K' or it holds that $\neg valid(o, pr, o', K')$, possibly because the domain and/or range of pr has been changed or specialization relationships between classes that involve (directly or indirectly) the domain and/or range of pr have been deleted in $S_{K'}$. These property instance triples have to be added to P_{K_Del} set because they either do not belong to $B_{K'}$ or belong to $Invalid(K')$.

If a property, pr, that appears in S_K and I_K, is removed in $S_{K'}$ and consequently the statements regarding its domain and range, to restore validity of $K' = (S_{K'}, I_K)$, we consider that the domain and range of pr, in K', is the class *Resource*. Consider, for instance, Fig. 9. The property instance triple $(o_1\ pr_1\ o_2) \in I_K$ leads to an invalid KB because pr_1 and thus, also its domain and range are deleted in K'. If we consider that the domain and range of pr_1, in K', is the class *Resource* then the validity of K' is restored.

Proposition 15 *The time complexity of Algorithm 2 is $O(|Inst_K|^2 * M^2 * (S^2 + |P_K|)$, where $S = max(|K|, |K'|)$ and $M = max(\{|C_K|, |Pr_K|, |C_{K'}|, |Pr_{K'}|\})$.*

We would like to note that Algorithm 2 is more general than Algorithm 1 in the sense that it can be applied even in the backwards compatible schema evolution case. However, Algorithm 1 has fewer steps and is more efficient.

7 Sequences of Transitions

In this Section, we investigate what happens in *sequences* of successive transitions.

Let $K_1 = (S_1, I_1)$ be a KB and let $S_2, \ldots, S_n$ be a sequence of new schema versions, resulting to KBs $K_i = (S_i, I_1)$, for $i = 2, \ldots, n$. Now consider the *one-step* migration of I_1 from the first (S_1) to the last (S_n) schema, i.e. consider the transition:

$$(\mathcal{C}_1, F_1, P_1) \rightsquigarrow (\mathcal{C}_n, F_n, P_n)$$

where $\mathcal{C}_1$ is based on S_1 and I_1, and suppose that $P_1 = \emptyset$[11].

[10] Note that $t \notin B_{K'}$, if the class c or property pr appearing in t have been deleted in K'.

[11] To keep notations simple, here and below we omit the subscripts K from the notations of $\mathcal{C}$, P and M.

Algorithm 2 *Produce_Possibilities*$_{NBC}(K, P_K, S_{K'})$
Input: a KB K, its set of possible instance triples P_K, and a new set of schema triples $S_{K'}$ s.t. $K' = (S_{K'}, I_K)$ is a (valid) KB
Output: the set of possible instance triples $P_{K'}$ of the KB K'

(1) $K' = (S_{K'}, I_K)$;
(2) $P_{K\_Add} = \emptyset$;
(3) $P_{K\_Del} = \emptyset$;
/* FOR CLASS INSTANCES: */
(4) $NC = C_{K'} \setminus C_K$; /* new classes appearing in K' */
/* PART A: New classes */
(5) For all $(c' \in NC)$ do /* for each new class */
(6) $P_{K\_Add} = P_{K\_Add} \cup \{(o\ type\ c') \mid o \in Inst_K \cap URI$,
$\forall\ c \in C_K$ s.t. $c' \leq^*_{cl} c$ it holds that $(o\ type\ c) \in (\mathcal{C}_i(K') \cup P_K)$, and
$(o\ type\ c') \notin \mathcal{C}_i(K')\}$;
/* PART B: Existing classes */
(7) For all $(c_1 \in C_K)$ do
/* Moving class instance triples from P_K to $F_{K'}$ due to Rule $R1$ and Rule $R4$. */
(8) $P_{K\_Del} = P_{K\_Del} \cup \{(o\ type\ c_1) \in P_K \mid c_2 \in C_K, c_1 \leq^*_{cl} c_2$, and
$(o\ type\ c_2) \notin (P_K \cup \mathcal{C}_i(K'))\}$;
/* PART C: Removing class instance triples from P_K, due to removed classes */
(9) For all $((\mathbf{o\ type\ c}) \in \mathbf{P_K})$ do
(10) If $\mathbf{c \notin C_{K'}}$ then $\mathbf{P_{K\_Del} = P_{K\_Del} \cup \{(o\ type\ c)\}}$;
/* FOR PROPERTIES: */
(11) $NP = Pr_{K'} \setminus Pr_K$; /* new properties appearing in K' */
/* PART D: New properties */
(12) For all $(pr' \in NP)$ do /* for each new property */
(13) $P_{K\_Add} = P_{K\_Add} \cup \{(o\ pr'\ o') \mid o \in URI$,
$valid(o, pr', o', K')$,
$\forall\ pr \in Pr_K$ s.t. $pr' \leq^*_{pr} pr$ it holds that$(o\ pr\ o') \in (\mathcal{C}_i(K') \cup P_K))$, and
$(o\ pr'\ o') \notin \mathcal{C}_i(K')\}$;
/* PART E: Existing properties */
(14) For all $(pr_1 \in Pr_K)$ do
/* Moving property instance triples from P_K to $F_{K'}$ due to Rule $R2$ and $R5$. */
(15) $P_{K\_Del} = P_{K\_Del} \cup \{(o\ pr_1\ o') \in P_K \mid pr_2 \in Pr_K, pr_1 \leq^*_{pr} pr_2$, and
$(o\ pr_2\ o') \notin (P_K \cup \mathcal{C}_i(K'))\}$;
/* PART F: Removing property instance triples from P_K due to removed properties, subject out of the domain, or object out of range */
(16) For all $((\mathbf{o\ pr\ o'}) \in \mathbf{P_K})$ do
(17) If $\mathbf{pr \notin Pr_{K'}}$ or $\neg$valid$(\mathbf{o, pr, o', K'})$ then
$\mathbf{P_{K\_Del} = P_{K\_Del} \cup \{(o\ pr\ o')\}}$;
/* Moving instance triples from P_K to $\mathcal{C}_i(K')$. */
(18) $P_{K\_Del} = P_{K\_Del} \cup (P_K \cap \mathcal{C}_i(K'))$;
(19) $P_{K'} = P_K \setminus P_{K\_Del}$;
(20) $P_{K'} = P_{K'} \cup P_{K\_Add}$;
(21) Return $P_{K'}$;

Now consider a *sequential migrations* scenario where I_1 is migrated to S_2, then to S_3, and so on, up to S_n. So, the scenario consists of the following sequence of $n-1$ transitions:

$$
\begin{aligned}
(\mathcal{C}_1, F_1, P_1) &\rightsquigarrow (\mathcal{C}'_2, F'_2, P'_2) \\
&\rightsquigarrow \ldots \\
&\rightsquigarrow (\mathcal{C}'_{n-1}, F'_{n-1}, P'_{n-1}) \\
&\rightsquigarrow (\mathcal{C}'_n, F'_n, P'_n)
\end{aligned}
$$

Let us now discuss the relationship between the outcome of the *one-step* migration, i.e. $(\mathcal{C}_n, F_n, P_n)$, with respect to the final outcome of the *sequential migrations*, i.e. with $(\mathcal{C}'_n, F'_n, P'_n)$. The main points are:

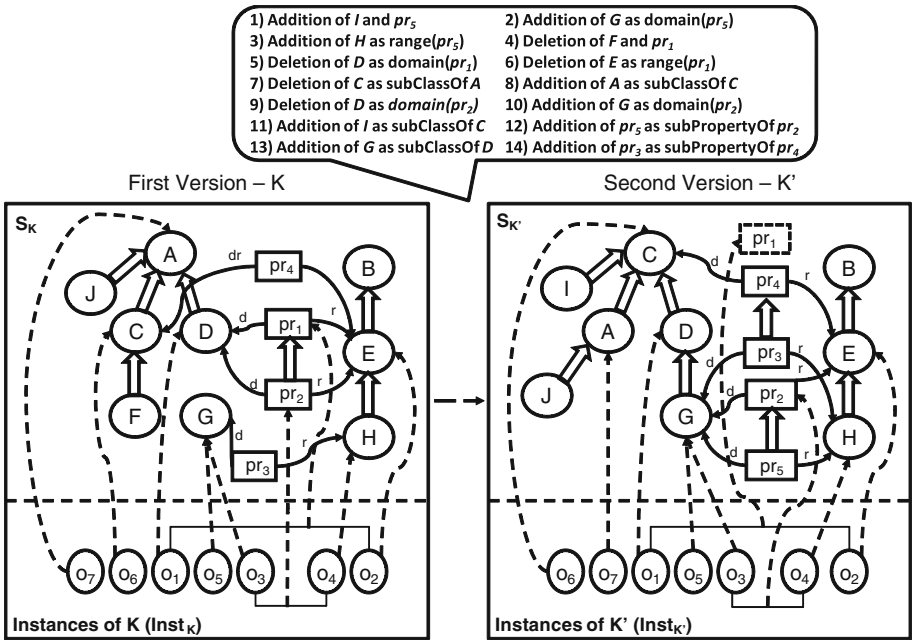

Fig. 9 Non-backwards compatible schema evolution

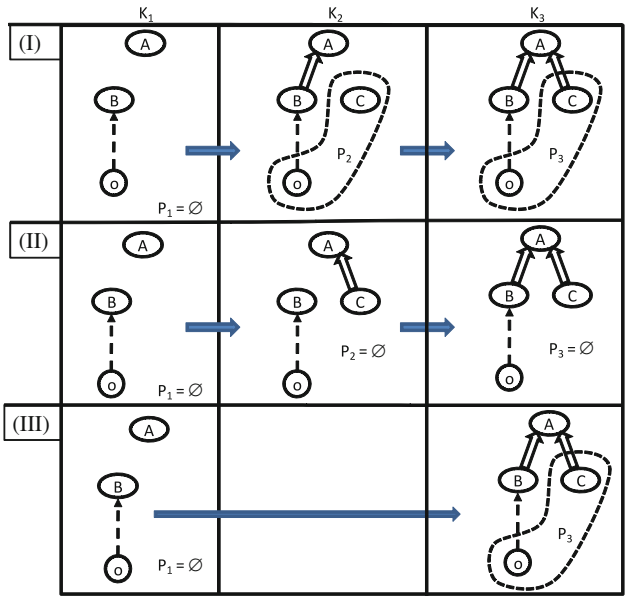

Fig. 10 Different sequences of transitions

(a) $\mathcal{C}_n = \mathcal{C}'_n$, i.e. the certain parts of the resulting partitions are the same. This is due to postulate Π1 of Definition 7.
(b) The rest parts of the partitions (i.e. F and P) can be different.

Point (b) can be made evident through a small example, like that of Fig. 10. The last row (III) shows the one step migration. The first and the second rows show two different sequential migrations that lead to the *same* final schema. Notice that the first sequence gives the same result with the one step migration, however the second does not.

We could say that this phenomenon is a kind of *"negative prejudgement, due to lack of the final (forthcoming) knowledge"*. In our case, the negative prejudgement is realized by postulate Π2.

[Suggested Policy]
If no user feedback is expected/given after a migration, then there is no need to compute or store the intermediate P_i. Instead, it is better to compute it between the first and the last schema, and only when needed (i.e. just before the curator starts the lifecycle process that we describe in the next section). In this way, we can bypass the "negative prejudgement" due to lack of the forthcoming knowledge.

Below, we present a proposition that indicates when the *one-step migration* gives the same result with the *sequential migrations* scenario.

Proposition 16 *Let $K_1 = (S_1, I_1)$ be a KB and let $S_2, \ldots, S_n$ be a sequence of backwards compatible schema versions, resulting to KBs $K_i = (S_i, I_1)$, for $i = 2, , \ldots, n$. Now consider the one-step migration of I_1 from the first (S_1) to the last (S_n) schema: $(\mathcal{C}_1, F_1, P_1) \rightsquigarrow (\mathcal{C}_n, F_n, P_n)$. Additionally, consider the sequential migrations scenario, where: $(\mathcal{C}_1, F_1, P_1) \rightsquigarrow (\mathcal{C}'_2, F'_2, P'_2) \rightsquigarrow \ldots \rightsquigarrow (\mathcal{C}'_n, F'_n, P'_n)$.*

If (a) for all $c_1, c_2 \in C_{K_1}$, it holds that $c_1 \leq^*_{cl} c_2$ in K_1 iff $c_1 \leq^*_{cl} c_2$ in K_n, and (b) for all $pr_1, pr_2 \in Pr_{K_1}$, it holds that $pr_1 \leq^*_{pr} pr_2$ in K_1 iff $pr_1 \leq^*_{pr} pr_2$ in K_n, then $(\mathcal{C}_n, F_n, P_n) = (\mathcal{C}'_n, F'_n, P'_n)$.

We would like to note that Proposition 16 does not hold in the case that the sequence of schema versions is *not*

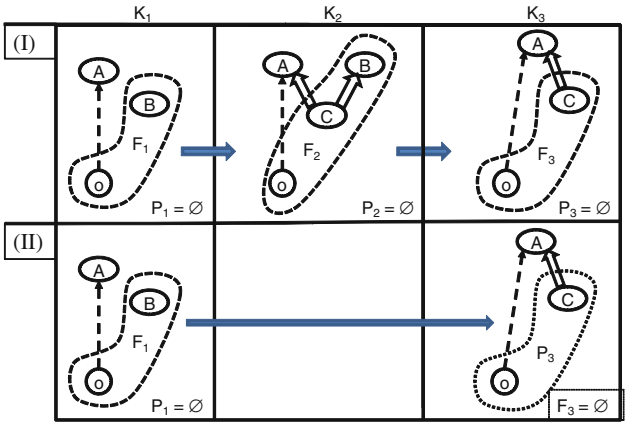

Fig. 11 Different sequences of non-backwards compatible transitions

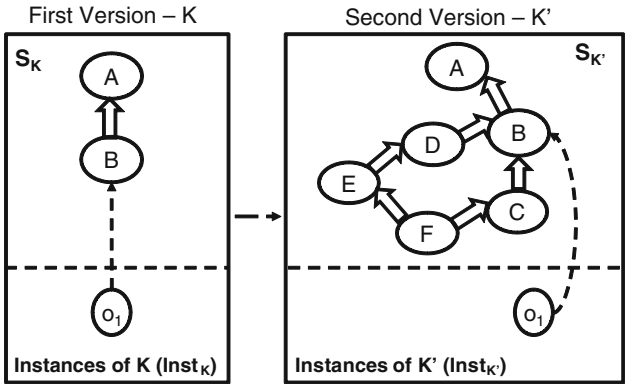

Fig. 12 Ranking possible class instance triples

backwards compatible (see Example 5 below). This happens because a class or property that exist in a schema version S_i may have been removed from the next schema version S_{i+1}.

Example 5 Consider Fig. 11. In KB K_1, there exist two classes `A` and `B` and `o` is an instance of `A`. Assuming that the *MSA* holds for K_1, it follows that $P_1 = \emptyset$ and $F_1 = \{(\texttt{o type B})\}$. In scenario (I), in KB K_2, a new class `C` is added, as well as the relationships $\texttt{C} \leq_{cl} \texttt{A}$ and $\texttt{C} \leq_{cl} \texttt{B}$. From Rule $R1$, it follows that $F_2 = \{(\texttt{o type B}), (\texttt{o type C})\}$. In K_3, class `B` is removed. Thus, $F_3 = \{((\texttt{o type C})\}$. In scenario (II), class `B` is removed and a new class `C` is added, along with the relationship $\texttt{C} \leq_{cl} \texttt{A}$. Then, since none of the rules $R1 - R5$ applies, it follows that $F_3 = \emptyset$ and $P_3 = \{(\texttt{o type C})\}$. Note that the non-backwards compatible transitions in scenarios (I) and (II) lead to different results.

8 Ranking Possibilities

Here, we discuss methods for ranking the computed possibilities for controlling the amount of possible information that is kept stored, and for aiding the interaction process that will be described later on.

If t is a triple in P_K or $\mathcal{C}_i(K)$, we could "score" t according to a *degree of certainty*. A naive approach would be to set $score(t) = 1$ if $t \in \mathcal{C}_i(K)$, and $score(t) = 1/2$ if $t \in P_K$. Below we introduce a more sophisticated model for ranking the possible triples. Consider an example, where $\texttt{PhD\_Student} \leq^*_{cl} \texttt{Postgraduate} \leq^*_{cl} \texttt{Student} \leq^*_{cl} \texttt{Person}$ and $\texttt{Manager} \leq^*_{cl} \texttt{Employee}$. Assume that `John` was originally classified to the class `Person`. We can say that `Student` and `Employee` are more probable classes for `John` than `Postgraduate`, `Manager`, and `PhD_Student`. To this end we propose an extension of P_K that we call *quantified* P_K, such that each triple is accompanied by a positive integer that is interpreted as follows: the *less* this value is the *more possible* the triple is.

Let $dist_{cl}(c \rightarrow c')$ be the length of the shortest path from class c to class c' comprised from $\leq_{cl}$ relationships (over the reflexive and transitive reduction of $\leq^*_{cl}$). If there is no path from class c to class c' comprised from $\leq_{cl}$ relationships then $dist_{cl}(c \rightarrow c') = \infty$. If $c \leq_{cl} c_1 \leq_{cl} c_2 \leq_{cl} c_3 \leq_{cl} c'$ and this is the shortest path from class c to class c' comprised from $\leq_{cl}$ relationships, then $dist_{cl}(c \rightarrow c') = 4$. For each element $(o\ type\ c)$ in P_K, the *quantified* P_K contains an element $((o\ type\ c), distClass(o, c))$, where:

$$distClass(o, c) = \min\{dist_{cl}(c \rightarrow c') \mid c' \in C_K \text{ and } o \in inst_K(c')\}$$

So, $distClass(o, c)$ is the shortest distance of c from one of the certain classes of o. A measure, similar to $distClass(o, c)$, appears in [23] for measuring the conceptual distance between two concepts.

Example 6 Consider Fig. 12. The second schema version $S_{K'}$ is backwards compatible with the first schema version S_K. The new set of classes from K to K' is {`C`, `D`, `E`, `F`}. So, according to Algorithm 1, the derived possible class instance triples are: $\{(o_1\ \texttt{type C}), (o_1\ \texttt{type D}), (o_1\ \texttt{type E}), (o_1\ \texttt{type F})\}$.

If we rank the possible class instance triples of the instance $\texttt{o}_1$, using the above formula, i.e. $distClass(o, c)$, where o corresponds to $\texttt{o}_1$ and c corresponds to one of the possible classes of $\texttt{o}_1$, we get the following quantified possible class instance triples:
$\{((\texttt{o}_1\ \texttt{type C}), 1), ((\texttt{o}_1\ \texttt{type D}), 1),$
$((\texttt{o}_1\ \texttt{type E}), 2), ((\texttt{o}_1\ \texttt{type F}), 2)\}$.

For example, note that in the case of the possible class instance triple $(\texttt{o}_1\ \texttt{type F})$, there are two paths from `F` to `B` (which is a certain class of $\texttt{o}_1$). The first one is $\texttt{F} \leq_{cl} \texttt{C} \leq_{cl} \texttt{B}$ and the second one is $\texttt{F} \leq_{cl} \texttt{E} \leq_{cl} \texttt{D} \leq_{cl} \texttt{B}$. Thus, $dist_{cl}(\texttt{F} \rightarrow \texttt{B})$ is 2. Additionally, there are two paths from `F` to `A` (which is also a certain class of $\texttt{o}_1$). The first one is

$\mathtt{F} \leq_{cl} \mathtt{C} \leq_{cl} \mathtt{B} \leq_{cl} \mathtt{A}$ and the second one is $\mathtt{F} \leq_{cl} \mathtt{E} \leq_{cl} \mathtt{D} \leq_{cl} \mathtt{B} \leq_{cl} \mathtt{A}$. Thus, $dist_{cl}(\mathtt{F} \rightarrow \mathtt{A})$ is 3. Therefore, $distClass(\mathtt{o}_1, \mathtt{F}) = min(\{2, 3\}) = 2$.

Similarly, let $dist_{pr}(pr \rightarrow pr')$ be the length of the shortest path from property pr to property pr' comprised from $\leq_{pr}$ relationships (over the reflexive and transitive reduction of $\leq^*_{pr}$, which is unique in our case because $\leq^*_{pr}$ is finite and acyclic). If there is no path from property pr to property pr' comprised from $\leq_{pr}$ relationships then $dist_{pr}(pr \rightarrow pr') = \infty$. If $pr \leq_{pr} pr_1 \leq_{pr} pr_2 \leq_{pr} pr_3 \leq_{pr} pr'$ and this is the shortest path from pr to pr' comprised from $\leq_{pr}$ relationships, then $dist_{pr}(pr \rightarrow pr') = 4$. We define an auxiliary property $pr_{\mathtt{dummy}}$ such that if $pr \in Pr_K$ and there is no $pr' \in Pr_K$ s.t. $pr \leq_{pr} pr'$ then we add $pr \leq_{pr} pr_{\mathtt{dummy}}$. For each element $(o\ pr\ o')$ in P_K, the *quantified* P_K contains an element $((o\ pr\ o'), distProperty(o, pr, o'))$, where:

$$distProperty(o, pr, o') = \begin{cases} \min\{dist_{pr}(pr \rightarrow pr') | pr' \in Pr_K, (opr'o') \\ \quad \in \mathcal{C}(K)\} \quad \text{if } \exists pr' \in Pr_K \text{ s.t. } (opr'o') \in \mathcal{C}(K) \\ dist_{pr}(pr \rightarrow pr_{\mathtt{dummy}}) \qquad \text{otherwise} \end{cases}$$

So, $distProperty(o, pr, o')$ is the shortest distance of pr from one of the properties pr' such that $(o\ pr'\ o') \in \mathcal{C}(K)$. In the case that there is no property pr' such that $(o\ pr'\ o') \in \mathcal{C}(K)$ then $distProperty(o, pr, o')$ is the distance of pr from $pr_{\mathtt{dummy}}$.

The $distClass(o)$ and $distProperty(o, o')$ values can be used for ranking the possible class instance triples and possible property instance triples, respectively.

9 Specificity Life Cycle Management

So far, we have seen principles and algorithms for computing possibilities. Now, we will focus on the management of these possibilities, specifically on the *exploitation* of the computed possibilities.

Suppose a number of instance descriptions that are migrated to a new schema version, and as a consequence our machinery has computed a set of possible instance triples P_K. If o is an instance, let us denote by $posTriples(o)$ all those triples that include o and belong to P_K. Suppose a system that for an instance o shows to the user a set of possible instance triples $U(o)$ such that $U(o) \subseteq posTriples(o)$. The user then decides whether he/she should add one or more than one of these to the certain knowledge base. After that we should also update P_K. Figure 4 (on the right) illustrates the proposed process.

In Fig. 4, on the left, we can see the current migration process, where the curator of a KB downloads its latest ontology version, he migrates the instance descriptions to that version, and then he manually tries to revise some of the migrated descriptions. On the right, we can see the proposed migration process. After the migration of the instance descriptions to the latest ontology version, the system computes the possible instance descriptions by executing either Algorithm 1 (when the new schema is backwards compatible with the previous one) or Algorithm 2 (when the new schema is not backwards compatible with the previous one). Then, an iterative procedure starts, where in each iteration, the curator can select a specific instance and then a ranked subset of its possible descriptions is displayed. The curator can accept or reject some or all of these possible descriptions (of the selected instance) and then the *eKB* (its certain and possible part) is updated, analogously. Below, we describe formally the updating of the certain and the possible part of the *eKB*.

What we have to ensure is that the update should: (a) respect the user's request, (b) reduce uncertainty based on what the user was prompted and decided, and (c) yield a valid *eKB* (that respects the constraint of Definition 10). To specify exactly the updating we need to introduce notations for the possible class instance triples and property instance triples of an instance o.

$$posTriples^{cl}(o) = \{(o\ type\ c) \mid (o\ type\ c) \in P_K\} \text{ // } cl \text{ : instance of class}$$
$$posTriples^{spr}(o) = \{(o\ pr\ o') \mid (o\ pr\ o') \in P_K\} \text{ // } spr \text{ : subject of property}$$
$$posTriples^{opr}(o) = \{(o'\ pr\ o) \mid (o'\ pr\ o) \in P_K\} \text{ // } opr \text{ : object of property}$$

At Sect. 3 we defined $SubTriples(\cdot)$. Now we define the *SuperTriples* of an instance triple, or a set of instance triples A, as follows:

$$SuperTriples((o\ type\ c)) = \{(o\ type\ c') \mid c \leq^*_{cl} c'\}$$
$$SuperTriples((o\ pr\ o')) = \{(o\ pr'\ o') \mid pr \leq^*_{pr} pr'\}$$
$$SuperTriples(A) = \bigcup_{t \in A} SuperTriples(t)$$

[Rejections].
Let o be an instance. If the system shows to the user all instance triples in $posTriples^t(o)$, where $t \in \{pr, spr, opr\}$, and he/she decides that none of these should be added then we should update P_K as follows: $P_K^{up} = P_K \setminus posTriples^t(o)$ where in P_K^{up}, up stands for *updated*. If the system had prompted to the user only a subset of the possibilities, say $U(o)$ (where $U(o) \subseteq posTriples^t(o)$), and the user had decided that none of these should be added then: $P_K^{up} = P_K \setminus SubTriples(U(o))$

Note that except from $U(o)$, the $SubTriples(U(o))$ should be removed from P_K due to (ii) of Definition 5. This ensures that not only $U(o)$ but also all possibilities which are more "specific" than those in $U(o)$ will be excluded. Note that

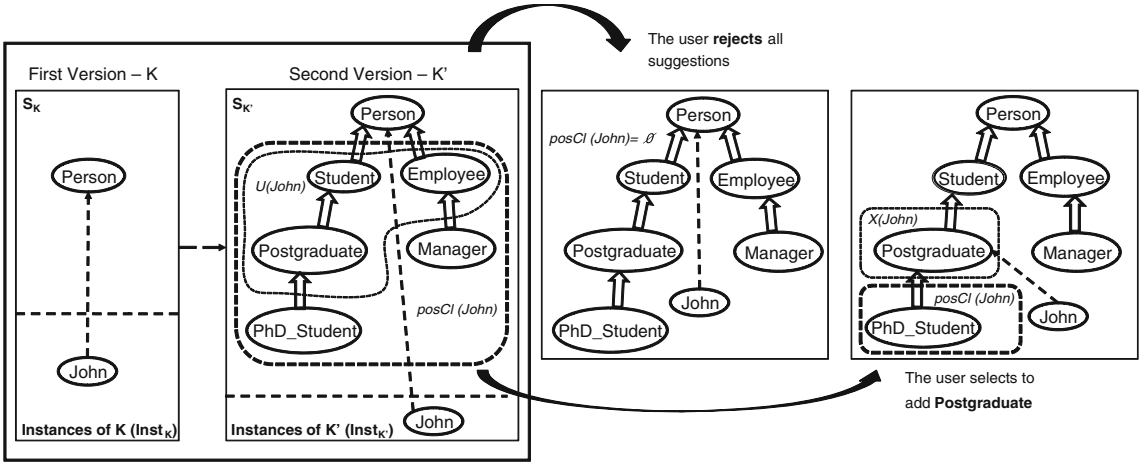

Fig. 13 Example of the Possibility Resolution Process

the instance triples in $U(o)$ and thus, the instance triples in $P_K \cap SubTriples(U(o))$, are actually moved to F_K^{up}.

[Acceptances].

If the user has selected some of the suggested possibilities, say $X(o)$, $(X(o) \subseteq U(o))$, to be added, then we should update the certain part of the new KB $K^{up} = K \cup X(o)$, as follows $\mathcal{C}(K^{up}) = \mathcal{C}(K) \cup SuperTriples(X(o))$ (see Proposition 17 below), and then update P_K, accordingly. The latter can be done as follows:

1. $P_K^{up} = P_K \setminus SuperTriples(X(o))$
2. $P_K^{up} = P_K^{up} \setminus SubTriples(U(o) \setminus SuperTriples(X(o)))$

The first step excludes from P_K also the newly entailed instance triples from K^{up}, i.e. $SuperTriples(X(o))$, (to satisfy the constraint of Definition 10). The second excludes from P_K^{up} the instance triples in $U(o) \setminus SuperTriples(X(o))$ and their specializations. Note that the instance triples in $U(o) \setminus SuperTriples(X(o))$, and thus the instance triples in $P_K \cap SubTriples(U(o) \setminus SuperTriples(X(o))$, are actually moved to F_K^{up}.

Proposition 17 *Let* $X \subseteq P_K$. *If* $K^{up} = K \cup X$ *then* $\mathcal{C}(K^{up}) = \mathcal{C}(K) \cup SuperTriples(X)$.

Example 7 Consider the scenario of Fig. 13. P_K and $posTriples^{cl}$(John) contain the triples: {(John type Employee), (John type Manager), (John type Student), (John type Postgraduate), (John type PhD_Student)} (shown enclosed in a dashed frame)[12]. If the system shows to the user all triples in $posTriples^{cl}$(John) and the user decides that none of these should be added, then P_K should be updated as follows: $P_K^{up} = P_K \setminus posTriples^{cl}(\texttt{John}) = \emptyset$.

[12] In the figure, $posCl(\texttt{John})$ denotes the possible classes of John.

Now suppose that the system had showed to the user only three of the five possible instance triples, such as: $U(\texttt{John})$ = {(John type Employee), (John type Student), (John type Postgraduate)}. If the user decides that he/she does not want to add any of these triples to the certain KB then P_K should be updated as follows:

$$P_K^{up} = P_K \setminus SubTriples(\{(\texttt{John type Employee}), (\texttt{John type Student})(\texttt{John type Postgraduate})\}) = \emptyset$$

However, if the user decides to add one of these three suggested triples, say the triple $X(\texttt{John}) = \{(\texttt{John type Postgraduate})\}$ then the KB has to be updated such that $K^{up} = K \cup \{(\texttt{John type Postgraduate})\}$. Note that $SubTriples(X(\texttt{John})) = \{(\texttt{John type Postgraduate}), (\texttt{John type PhD\_Student})\}$ and $SuperTriples(X(\texttt{John})) = \{(\texttt{John type Postgraduate}), (\texttt{John type Student}), (\texttt{John type Person})\}$.

The possible part P_K has to be updated as follows:

1. $P_K^{up} = \{(\texttt{John type Employee}), (\texttt{John type Manager}), (\texttt{John type Student}), (\texttt{John type Postgraduate}) (\texttt{John type PhD\_Student})\} \setminus SuperTriples(X(\texttt{John}))$
   $= \{(\texttt{John type Employee}), (\texttt{John type Manager}), (\texttt{John type PhD\_Student})\}$
2. $P_K^{up} = P_K^{up} \setminus SubTriples(U(\texttt{John}) \setminus SuperTriples(X(\texttt{John})))$
   $= P_K^{up} \setminus SubTriples(\{(\texttt{John type Employee})\})$
   $= \{(\texttt{John type PhD\_Student})\}$

The updated KB K, i.e. K^{up}, and the updated P_K, i.e. P_K^{up}, can now be given as input to Algorithm 1, in the case that

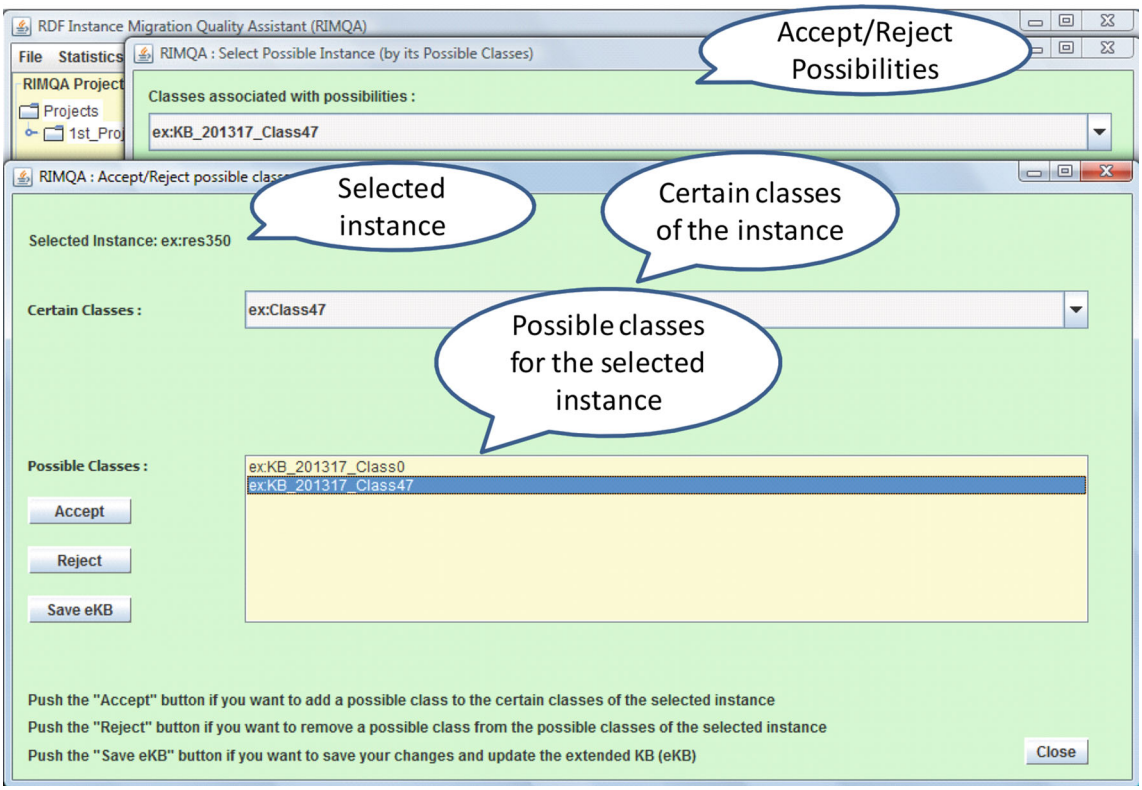

Fig. 14 Selected screenshot of RIMQA

a new set of schema triples $S_{K'}$ is available, for generating the new $P_{K'}$. Note that the set of instance triples that is migrated from S_K to $S_{K'}$ is now $I_{K^{up}}$.

The ranking methods defined at Sect. 8 can aid the interaction, e.g. at the example of Fig. 13, the system will suggest first Student and Employee as possible classes of John. If the user will not select any of these then P_K will be updated and Postgraduate, Manager, and PhD_Student will never be suggested as possible classes (for the instance John).

However, the important point is that if the curation process is followed and the curator accepts/rejects the migration-related uncertainties, then the possible part of the KB becomes empty, i.e. no extra storage is required.

10 Implementation and Application Issues

We have implemented a proof-of-concept prototype, called system (**R**DF **I**nstance **M**igration **Q**uality **A**ssistant) which supports the entire lifecycle that we have proposed[13] selected screenshot of the system is shown in Fig. 14. Below we report some experimental results for demonstrating the feasibility of the approach (Sect. 10.1), we introduce methods for compacting the representation of possibilities (Sect. 10.2), and finally we discuss applicability issues (Sect. 10.3).

10.1 Experimental Evaluation

We have conducted an experimental evaluation whose objectives is to (a) measure the size in triples of the computed $P_{K'}$, and (b) measure the time required to compute $P_{K'}$ for investigating the applicability of this method to large datasets. Regarding datasets and measurements, we adopt the following methodology. For each dataset we get a sequence of schema versions, specifically a sequence of the form $S_{K_0}, \ldots, S_{K_n}$, where each S_{K_i} is a set of schema triples, and a set of instance triples I_K w.r.t. the first version of the schema, i.e. S_{K_0}. Subsequently, we migrate I_K to each of the subsequent versions of the schema and for each one we compute the corresponding P_{K_i}. Specifically, we do the migrations $S_{K_{i-1}} \rightarrow S_{K_i}$ for all $i = 1 \ldots n$. We use two scenarios: in the first, we consider that $P_{K_{i-1}} = \emptyset$ and thus the computation of P_{K_i} depends only on the current migration, i.e. $S_{K_{i-1}} \rightarrow S_{K_i}$, while in the second $P_{K_{i-1}}$ has been specified from the migration $S_{K_{i-2}} \rightarrow S_{K_{i-1}}$ (or former migrations).

[13] More information about the prototype is available through its web page: http://www.ics.forth.gr/isl/RIMQA.

Table 2 $|C_i|$, $|Pr_i|$, $|I_i|$ and $|S_i|$ for each i Music Ontology version

	Versions of Music Ontology (S_i)							
	v.7 .08 .2007	v.10 .08 .2007	v.12 .08 .2007	v.18 .09 .2007	v.6 .12 .2007	v.28 .07 .2008	v.28 .10 .2008	v.13 .02 .2010
$\|C_i\|$	113	94	93	93	94	86	124	95
$\|C_i\| - \|C_{i-1}\|$		−19	−1	0	1	−8	38	−29
$\|Pr_i\|$	147	167	167	167	174	160	160	183
$\|Pr_i\| - \|Pr_{i-1}\|$		20	0	0	7	−14	0	23
$\|I_i\|$	26	26	26	26	26	26	26	26
$\|S_i\|$	1059	1266	1266	1259	1269	1115	1153	1302
$\|S_i\| - \|S_{i-1}\|$		207	0	−7	10	−154	38	149

Table 3 $|P_{K'}|$ Sizes for Music Ontology migrations $|P_K| = 0$

$\|P_{K'}\|$ changes at each part of Alg. 2	$\|P_{K'}\|$ Sizes (for migrations where $P_K = \emptyset$)						
	v.7 .08 .2007 → v.10 .08. 2007	v.10. 08. 2007 → v.12 .08.2007	v.12 .08. 2007 → v.18 .09.2007	v.18 .09. 2007 → v.6 .12.2007	v.6 .12.2007 → v.28 .07. 2008	v.28 .07. 2008 → v.28 .10.2008	v.28 .10. 2008 → v.13 .02. 2010
A	0	0	0	+21	+34	+289	+153
D	0	0	0	0	0	0	0
Total	0	0	0	21	34	289	153

The implementation is written in Java using the SWKM[14] (Semantic Web Knowledge Middleware) main memory model[15]. One important implementation detail is that we *do not have to compute the closure* of any of the involved KBs. Instead we check whether a particular triple belongs to the closure and this is done efficiently by exploiting the labeling schemes [5] supported by SWKM.

10.1.1 Real Data Set

We used the RDF/S versions of Music Ontology[16] which is a formal framework for dealing with music-related information on the Semantic Web, including editorial, cultural and acoustic information. In our experiments, we used the following successive versions: v.7.08.2007, v.10.08.2007, v.12.08.2007, v.18.09.2007, v.6.12.2007, v.28.07.2008, v.28.10.2008 and v.13.02.2010. Since each version is not backwards compatible with the previous ones (although it is migration compatible), Algorithm 2 is used.

For each version, Table 2 shows the number of classes (i.e. $|C_i|$), the new classes added from S_K to $S_{K'}$ (i.e. $|C_i| - |C_{i-1}|$), the number of properties (i.e. $|Pr_i|$), the new properties added from S_K to $S_{K'}$ (i.e. $|Pr_i| - |Pr_{i-1}|$), the number of explicit instance triples (i.e. $|I_K|$) that are migrated and the number of explicit schema triples (i.e. $|S_K|$). The last line shows the size difference in schema triples between S_K and $S_{K'}$. We can see that version v.18.09.2007 has fewer schema triples than version v.12.08.2007 and version v.28.07.2008 has fewer schema triples than version v.6.12.2007. This means that a number of deletions has taken place from v.12.08.2007 to v.18.09.2007 and from v.6.12.2007 to v.28.07.2008, accordingly.

Number of possible triples. Table 3 shows the number of possible triples (i.e. $|P_{K'}|$) produced during the migration from S_K to $S_{K'}$, assuming that $P_K = \emptyset$, while in Table 4 we consider that $P_K \neq \emptyset$ for each migration from S_K to $S_{K'}$. Specifically in Table 4, P_K is the one derived by the previous migration. The last lines of these tables show the final $|P_{K'}|$ for each migration $S_K \rightarrow S_{K'}$. As we can see in Table 3, during the migrations v.7.08.2007 → v.10.08.2007, v.10.08.2007 → v.12.08.2007, and v.12.08.2007 → v.18.09.2007, Algorithm 2 does not return any possible triples. Moreover, note that the only possible triples that are produced, are those after the execution of part A of Algorithm 2, i.e. after the update of P_K as a consequence of the new classes added from S_K to $S_{K'}$. Note that in Table 3, only the parts A and D of Algorithm 2 are shown, because the rest parts do not reduce the possible triples, since

[14] http://athena.ics.forth.gr:9090/SWKM.

[15] All experiments were carried out in an ordinary laptop with processor Pentium(R) Dual-Core CPU T4200 @2.0 Ghz, 2 GB Ram, running Windows Vista.

[16] http://www.musicontology.com.

Table 4 $|P_{K'}|$ Sizes for Music Ontology migrations $|P_K| \neq 0$

$\|P_{K'}\|$ changes at each part of Alg. 2	$\|P_{K'}\|$ Sizes (for migrations where $P_K \neq \emptyset$)		
	v.6.12.2007 → v.28.07.2008 P_K (v.18.09.2007 → v.6.12.2007)	v.28.07.2008 → v.28.10.2008 P_K (v.6.12.2007 → v.28.07.2008)	v.28.10.2008 → v.13.02.2010 P_K (v.28.07.2008 → v.28.10.2008)
$\|P_K\|$	21	34	289
A	+34	+289	+153
B	0	0	0
C	0	−17	−289
D	0	0	0
E	0	0	0
F	0	0	0
Line 18	0	0	0
Total	55	306	153

Table 5 Execution times (in ms for Music Ontology migrations for $|P_K| = 0$

Times for parts of Alg. 2	$P_{K'}$ Execution times (for migrations where $P_K = \emptyset$)						
	v.7 .08.2007 → v.10 .08.2007	v.10 .08.2007 → v.12 .08.2007	v.12 .08.2007 → v.18 .09.2007	v.18 .09.2007 → v.6 .12.2007	v.6 .12.2007 → v.28 .07.2008	v.28 .07.2008 → v.28 .10.2008	v.28 .10.2008 → v.13 .02.2010
A	11.0	4.7	0.6	3.3	4.2	88.5	23.8
D	4.7	0.1	0.1	3.9	3.4	1.3	4.2
Total	15.7	4.8	0.7	7.2	7.6	89.8	27.7

$P_K = \emptyset$. In Table 4, we can see that possible triples are produced again only by part A and we also have deletions by part C of Algorithm 2.

Execution times. Tables 5 and 6 show the execution times, corresponding to the scenarios of Tables 3 and 4, respectively. We can see that the migration takes just some milliseconds and mainly depends on the number of the new classes (part A) added from S_K to $S_{K'}$.

10.1.2 Synthetic Data Set

To conduct experiments over larger datasets and backwards compatible ontologies, we created and used one synthetic data set. Specifically, using the synthetic KB generator described in [27], we created a KB, v.K1, with 100 classes and 300 properties. To obtain a schema whose features resemble those of real ones, the subsumption relation follows a power law distribution. Specifically and accordingly to the metrics used in [28], we set the power-law exponent to 0.5 for the total-degree VR[17] function of the property graph and to 1.7 for the PDF[18] function of the descendants distribution. The depth of the class subsumption hierarchy in the schema is 7. Subsequently, we created three subsequent schemas of v.K1, namely v.K2, v.K3 and v.K4; v.K2 was derived by adding to v.K1 14 new classes as specializations to randomly selected leaf classes of v.K1 and similarly 10 new properties as specializations to randomly selected properties of v.K1; v.K3 was derived by adding to v.K2 14 new classes as specializations to randomly selected leaf classes of v.K2 (where 11 of them existed also in v.K1) and similarly 10 new properties as specializations to randomly selected properties of v.K2 (where 9 of them existed also in v.K1); v.K4 was derived by adding to v.K3 14 new classes as specializations to randomly selected leaf classes of v.K3 (where 13 of them existed also in v.K1) and similarly 10 new properties as specializations to randomly selected properties of v.K3 (where 9 of them existed also in v.K1).

For each class of v.K1 we created 100 instances, while for each property of v.K1 we created 10 property instance triples, among randomly selected instances of the corresponding

[17] VR stands for Value vs Rank (it measures the relationship between the i^{th} biggest value and its rank i, assuming a descending order).

[18] PDF stands for Probability Density Function.

Table 6 Execution times (in ms) for Music Ontology migrations for $|P_K| \neq 0$

Times for parts of Alg. 2	$P_{K'}$ Execution times (for migrations where $P_K \neq \emptyset$)		
	v.6.12.2007 → v.28.07.2008 P_K (v.18.09.2007 → v.6.12.2007)	v.28.07.2008 → v.28.10.2008 P_K (v.6.12.2007 → v.28.07.2008)	v.28.10.2008 → v.13.02.2010 P_K (v.28.07.2008 → v.28.10.2008)
A	3.5	167.0	210.9
B	0.6	0.5	1.8
C	0.1	0.5	4.0
D	0.8	0.1	4.0
E	0.2	0.2	0.3
F	0.1	0.1	0.1
Line 18	0.5	0.6	4.6
Total	5.8	169.0	225.7

Table 7 $|C_i|$, $|Pr_i|$, $|I_i|$ and $|S_i|$ for each i version of the Synthetic data set

	Versions of synthetic data (S_i)			
	v.K1	v.K2	v.K3	v.K4
$\|C_i\|$	101	115	129	143
$\|C_i\| - \|C_{i-1}\|$		14	14	14
$\|Pr_i\|$	289	299	309	319
$\|Pr_i\| - \|Pr_{i-1}\|$		10	10	10
$\|I_i\|$	3790	3790	3790	3790
$\|S_i\|$	1166	1219	1273	1338
$\|S_i\| - \|S_{i-1}\|$		53	54	65

Table 8 $|P_{K'}|$ Sizes for synthetic data migrations $|P_K| = 0$

$\|P_{K'}\|$ changes at each part of Alg. 1	$\|P_{K'}\|$ Sizes (for migrations where $P_K = \emptyset$)		
	v.K1 → v.K2	v.K2 → v.K3	v.K3 → v.K4
A	+ 1290	+ 230	+ 150
C	+ 97	+ 87	+ 86
Total	1387	317	236

domain and range classes. Table 7 shows the features of these schemas and the number of instance triples.

Number of possible triples. Since each version is backwards compatible with the previous ones, Algorithm 1 is used. Table 8 shows the number of possible triples (i.e. $|P_{K'}|$) produced during the migration from S_K to $S_{K'}$, assuming that $P_K = \emptyset$, while in Table 9 we consider that $P_K \neq \emptyset$ for each migration from S_K to $S_{K'}$. Specifically in Table 9, P_K is the one derived by the previous migration. The last lines of these tables show the final $|P_{K'}|$, for each migration $S_K \rightarrow S_{K'}$.

Note that in Table 8, migration v.K2 → v.K3 and v.K3 → v.K4 produce fewer possible class instance triples than v.K1 → v.K2. This is because some of the added classes from v.K2 to v.K3 (or from v.K3 to v.K4 respectively) are subclasses of classes added from v.K1 to v.K2 (or from v.K1 to v.K2 and from v.K2 to v.K3 respectively), which have no instances (although the rest classes have instances).

Execution times. Tables 10 and 11 show the execution times, corresponding to the scenarios of Tables 8 and 9, respectively. We can observe times that range from 4 s to 7 min and the cost mainly depends on the number of new classes (part A) and on the number of new properties (part C).

10.1.3 Scalability Issues and Optimizations

We have seen that parts A and C of Algorithm 1 and parts A and D of Algorithm 2 require most of the time. Below, we analyze their costs. Specifically, in part A of Algorithm 1

Table 9 $|P_{K'}|$ Sizes for synthetic data migrations $|P_K| \neq 0$

$\|P_{K'}\|$ changes at each part of Alg. 1	$\|P_{K'}\|$ Sizes (for migrations where $P_K \neq \emptyset$)	
	v.K2 →v.K3 P_K (v.K1 → v.K2)	v.K3 → v.K4 P_K (v.K2 → v.K3)
$\|P_K\|$	1387	317
A	+1250	+150
B	0	0
C	+97	+86
D	0	0
Line 14	0	0
Total	2734	553

Table 10 Execution times (in seconds) for synthetic data migrations for $|P_K| = 0$

Times for parts of Alg. 1	$P_{K'}$ Execution times (for migrations where $P_K = \emptyset$)		
	v.K1 → v.K2	v.K2 → v.K3	v.K3 → v.K4
A	1.3	3.0	1.7
C	4.8	3.7	2.6
Total	6.1	6.7	4.3

Table 11 Execution times (in seconds) for Synthetic Data migrations for $|P_K| \neq 0$

Times for each part of Alg. 1	$P_{K'}$ Execution times (for migrations where $P_K \neq \emptyset$)	
	v.K2 → v.K3 P_K (v.K1 → v.K2)	v.K3 → v.K4 P_K (v.K2 → v.K3)
A	284.7	52.2
B	1.4	0.03
C	111.9	38.3
D	0.3	0.2
Line 14	0.04	0.01
Total	398.5	90.74

and Algorithm 2, for each *new* class or property, we have to make an iteration whose cost is equal to the number of elements of $Inst_K$ multiplied by the number of the superclasses of the new class/property, multiplied by the cost of performing two lookups in closures. Specifically, the lookups, for part A, are the following: $(o\ type\ c) \in (\mathcal{C}_i(K') \cup P_K)$ and $(o\ type\ c') \notin \mathcal{C}_i(K')$, while the lookups for part C are: $(o\ pr\ o') \in (\mathcal{C}_i(K') \cup P_K)$ and $((o\ pr'\ o') \notin \mathcal{C}_i(K')$.

Therefore, we can say that the cost essentially depends on $Inst_K$, the number of new classes/properties, their depth (specifically their superclasses) and the cost of lookup in closures and in P_K.

First, note that the lookups can be performed fast if the involved closures are materialized and indexed. This is a common practice that is followed for supporting efficient query answering (this means that it is not unrealistic to expect that $\mathcal{C}_i(K')$ has been produced and stored). In the reported times we had not materialized the closures.

Various other optimizations can be devised and applied. For instance, for each $c' \in NC$, we can get each direct superclass $c \in C_K$ of c' and if c has no certain or possible instances then we can ignore c'. The gain is that we avoid unnecessary scans at $Inst_K$.

Even if we cannot ignore a new class c', we can ignore instances $o \in Inst_K$ if there is a superclass $c \in C_K$ of c' such that $(o\ type\ c) \notin \mathcal{C}_i(K') \cup P_K$. In this case, not all superclasses $c \in C_K$ of c' need to be checked for satisfying the condition $(o\ type\ c) \in \mathcal{C}_i(K') \cup P_K$.

The above optimizations are just indicative. One could even change the flow of control of part A. For instance, instead of scanning the entire $Inst_K$ in line (6) of Algorithms 1 and 2 (part A), one could scan only the URIs of the instances

of the classes which are superclasses of the new classes. Specifically, instead of the expression $o \in Inst_K \cap URI$, we can have $o \in I_A \cup I_B$, where I_A and I_B are computed as follows: at first, we compute the superclasses of the new classes, i.e. $SupsOfNewClasses = \bigcup_{c' \in NC}\{c \mid c' \leq^*_{cl} c\}$, and then we get their instances in $C_i(K')$ and P_K, i.e. $I_A = \{o|(o\ type\ c) \in C_i(K')$ and $c \in SupsOfNewClasses\}$, $I_B = \{o|(o\ type\ c) \in P_K$ and $c \in SupsOfNewClasses\}$.

The gain from the above change is that now the cost of part A of Algorithms 1 and 2 does not depend on $Inst_K$ but on $|NC|$ multiplied by the average number of instances that a class has plus the average number of possible instances of a class. Of course, we have to consider the cost of computing $SupsOfNewClasses$, I_A, and I_B, which however does not increase the theoretical complexity. Note that such optimization is not needed for line (11) of Algorithm 1 (part C) and line (13) of Algorithm 2 (part D) because $valid(o, pr', o', K')$ should hold.

Other optimizations can be designed depending on the implementation context, e.g. if a triple store is used then the set $SupOfNewClasses$ can be computed by a single SPARQL query. The same is true for the sets I_A and I_B.

What we have seen in our experiments is that the computation of possibilities after a migration, using an ordinary laptop and *without* any special optimization, takes at most 7 min for the KBs in our datasets. With optimizations like those that we have sketched before, that could be significantly reduced.

Finally, we should say that since migration is not an every day task, the computation of possibilities after a migration is not expected to have any significant scalability problem.

10.2 A Compact Representation for Possibilities

Even if our approach does not require storing any kind of negative information, one may wonder whether the proposed approach is still prohibitively expensive, in terms of space, to apply in large datasets. Two remarks are in order. The first is that the lifecycle process reduces the possibilities that have to be kept. The second point is that we can reduce the size required for the possible instance triples by exploiting various properties that hold.

At first note that if two classes, say c_1 and c_2, are possible classes for an instance o and it holds $c_1 \leq^*_{cl} c_2$ then all classes between, i.e. all c' such that $c_1 \leq^*_{cl} c' \leq^*_{cl} c_2$, are also possible classes for o [recall Lemma 2(1)]. This allows devising storage representations based on *intervals* over the reflexive and transitive reduction of the $\leq^*_{cl}$ relation. For example, if it holds $c_1 \leq_{cl} c_2 \leq_{cl} c_3 \leq_{cl} \cdots \leq_{cl} c_{10}$ and all of them are possible classes for o then we can represent them by the interval $[c_1, c_{10}]$.

If for a given instance, o, there are several intervals having a common end (it is more probable to have a common right end), then we could save more space by adopting a more compact representation, e.g. the intervals $[c_1, c_{10}]$ and $[c_2, c_{10}]$ can be represented by $[\{c_1, c_2\}, c_{10}]$. Moreover, if we have a compact representation of the form $[\{c_1, \ldots, c_k\}, c]$ and $\{c_1, \ldots, c_k\}$ are the leaves of the hierarchy rooted at c then we can even omit their representation and adopt a more declarative method, like $[*, c]$, meaning that all subclasses of c are possible classes for o. In case where a class c is a possible class of an instance o but there are not subclasses of c or superclasses of c that are possible classes of o then the corresponding compact representation is $[c, c]$, indicated by a point interval $[c]$, for short.

We can follow an analogous approach for property instances. If two property instance triples, say $(o\ pr_1\ o')$ and $(o\ pr_2\ o')$, are possible property instance triples for two instances o and o' and it holds $pr_1 \leq^* pr_2$ then, for all properties pr' such that $pr_1 \leq^*_{pr} pr' \leq^*_{pr} pr_2$, $(o\ pr'\ o')$ is also a possible property instance triple for o and o' [recall Lemma 2(2)]. This again allows compacted representations based on intervals over $\leq_{pr}$. For example, if it holds $pr_1 \leq_{pr} pr_2 \leq_{pr} pr_3 \leq_{pr} \cdots \leq_{pr} pr_{10}$ and $(o\ pr_i\ o')$, for all $i \in \{1, \ldots, 10\}$, is a possible property instance triple for o and o', then we can represent these possible instance triples by the interval $[pr_1, pr_{10}]$. The same storage policy (intervals with the same right end), as in class instance triples above, can be followed in the case of property instance triples.

Figure 15(left) illustrates a data structure for the compact representation, which follows an *object-centered storage policy* (beneficial for the requirements of the life cycle management) regarding classes. As we can see, on the left there is a list of all the instances, lexicographically ordered. In the middle, there is a list of pointers each pointing to an interval on the right part, where all the intervals of the compact representation of possibilities are found. Note that every instance o on the left points to a consecutive list of pointers in the middle and thus, to a list of intervals on the right. For example, instances o_1 and o_2, in Fig. 15(left), point to the same intervals, i.e. the first interval `[*,A]` and the second interval `[{B,C,D},E]`. Figure 15(right) illustrates the corresponding data structure for properties. Note that two or more pairs of instances may have the same compact representation. For example, (o_1, o_2) and (o_1, o_3), in Fig. 15(right), point to the same two intervals, i.e. `[a]` and `[{b,c,d},e]`.

We should clarify that the above compression technique is orthogonal and complementary to the various techniques which have been proposed (e.g. [7,17]) for compressing RDF triples by using ids for each (subject-predicate-object) element of the triples. Specifically, the internal-based method exploits Lemmas 2(1) and 2(2), something that would not be possible with a generic approach for compression. However, the space required by the interval method could be further reduced by using ids for the triple elements.

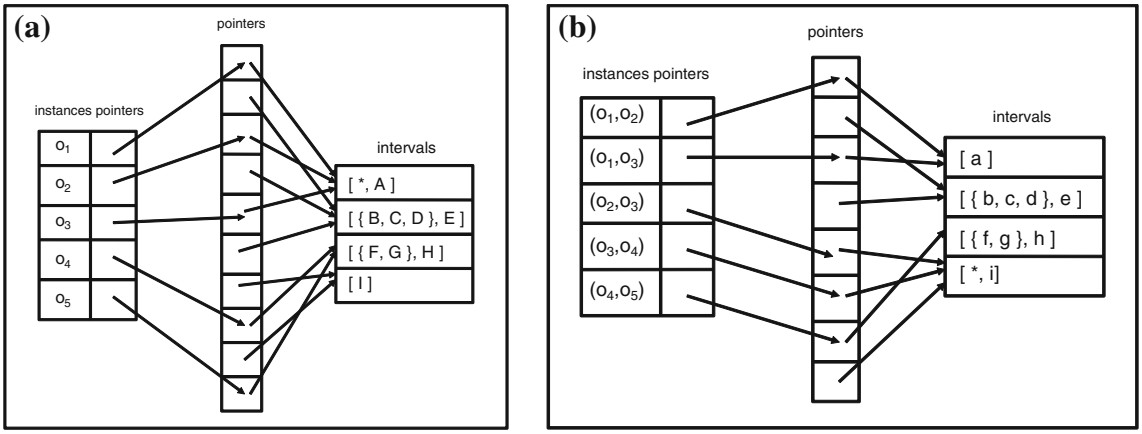

Fig. 15 Object-centered compact storage policy for possible class instance triples (left), and possible property instance triples (right)

The main advantage of adopting the compact representation is the space saving that we can achieve. However, if we execute Algorithms 1 and 2 using the compact representation of possibilities, we can see that the cost of looking up a specific possible instance triple in a compact representation is higher than in an explicit representation of all possibilities. Specifically, to decide whether a class instance triple (o *type* c) belongs to P_K requires locating the instance o and checking every interval which is pointed to by o.

Example 8 Consider Fig. 15. Suppose that we want to find out if the class instance triple (`o1 type K`) belongs to the compact representation of possibilities. Assume that `C` $\leq^*_{cl}$ `K` and `K` $\leq^*_{cl}$ `E` holds. At first, we find `o1` from the list of instances and then we check the intervals that `o1` points to, i.e. `[*,A]` and `[{B,C,D},E]`. Thus, we first check if `K` $\leq^*_{cl}$ `A` holds. Since this is not true, we check if `K` $\leq^*_{cl}$ `E` and if (`B` $\leq^*_{cl}$ `K` or `C` $\leq^*_{cl}$ `K` or `D` $\leq^*_{cl}$ `K`). Since these conditions hold, (`o1 type K`) is a possible class instance triple.

Let us now comment the time complexity of looking up an instance triple. Let $P_K^{explicit}$ denote an explicit representation of all possibilities ordered lexicographically, and $P_K^{compact}$ its compacted version. The time complexity of looking up an instance triple in $P_K^{explicit}$ (using binary search) in is $O(log_2(|P_K|))$. In $P_K^{compact}$ the cost depends on (a) the number of intervals associated with o, and (b) the cost of subsumption checking. The cost of the later is in $O(|\leq_{cl}|)$ if a plain graph structure is used (and no special index). However, given that most RDF triple stores currently use *labeling schemes* for encoding transitive subsumption relationships, subsumption checking can be performed in $O(1)$ (see [14,30]).

10.3 Application Issues

The proposed technique can be applied to any ontology-based repository. Just indicatively, CIDOC CRM[19] is one ontology (expressed in RDF/S) which is used by several ongoing EU projects, and it is curated (i.e. extended) by an authority (community) that is different from the various communities, curators of repositories, or simple users who keep creating instance descriptions with respect to that ontology. In practice, whenever a new version appears, the available instance descriptions are migrated to that version, and it is worth noting that this ontology has been revised at least five times the last 2 years (one recent version is described at [26]).

Collaborative Curation (or crowdcurating). The proposed technique can also be applied in a *crowdsourcing* setting, where several curators could interact in parallel after the migration of a dataset for curating its specificity. Specifically, each curator could carry out the steps in the area marked ≪interactive≫ in the right part of Fig. 4. A server stores the extended KB and whenever a curator accepts or rejects one possibility, the changes are applied in the KB of the server in the context of a transaction. In this way, no inconsistency will arise.

[Applicability to Other Domains]

Although we confine ourselves to RDF/S, the results of our work can be applied to any object-oriented conceptual modeling approach that supports classes, inter-class associations, specialization/generalization hierarchies (among classes and among inter-class associations) and instantiation. Below we

[19] CIDOC CRM (ISO 21127) is a core ontology describing the underlying semantics of data schemata and structures from all museum disciplines and archives (its RDF representation contains 78 classes and 250 properties from which 7 are literal-valued) (available from http://www.cidoc-crm.org/).

sketch an application in object-oriented software engineering.

Software Engineering. Our approach can be used also in object-oriented software engineering for upgrading libraries. Commonly, custom software relies on several libraries usually bundled in the form of jars. Most libraries evolve over time and their versions in most cases are backwards compatible. A new version of a library usually offers new subclasses of existing classes which provide improved/diversified functioning while respecting the ADT (Abstract Data Type) of the superclass. If the new version of a library is backwards compatible with the previous version, replacing the old version with the new version is enough for upgrading a software that depends on that library. However, this does not allow exploiting the new subclasses of the library: the user has to refer to textual descriptions and release notes in order to identify the new classes/features. An IDE (Integrated Development Environment) could adopt our approach for aiding the developer to upgrade his code. Specifically, it could be used for providing suggestions for refinements for those classes that are used by the code, i.e. those objects that instantiate library classes, and this can be done gradually by the lifecycle process.

We should also mention that RDF has been proposed as a data structure for software engineering, specifically for expressing software structure and dependencies[20].

As a brief and very rough example, and assuming a Java library, each Java class corresponds to an RDF class, each public instance variable of a class A with name v and type B corresponds to a property v with $domain(v) = A$ and $range(v) = B$, each object o that instantiates a class A corresponds to a class instance triple ($o\ type\ A$), etc.

10.4 Feedback from Related Communities and Users

We have already showed that the proposed method is feasible in terms of efficiency, and a tool that implements this approach is already available.

Regarding applicability, we have already communicated this work to the organizations participating to the NoE *APARSEN* (Alliance Permanent Access to the Records of Science in Europe, FP7, Proj. No 269977, 2011–2014), and it is referred to the deliverable [20]. The ultimate objective is to plan with the interested partners, a pilot application in an operational setting. We do not foresee any problem or unexpected issues, since this work actually proposed a process for making the curation easy and gradual, and the implemented tool provides the offered functionality in various different ways: starting from a particular instance, class or property, etc. However, its application in a real setting, will probably require adapting it to the processes and systems of the particular data provider/curator. Of course, the developed tool is not a polished end-user product, so the aspect of user interface usability and user-experience has not been considered in depth.

Our experience from a recent national conference of academic libraries[21], is that such organizations (libraries, national archives) are not aware of the problem, i.e. of the gradual loss of specificity that happens after each migration. The organizations that participated to that event understood directly the problem and they were interested in learning more about this. So, the key issue is to make such organizations aware of, and then to agree to undertake the responsibility for curating the quality of their metadata.

However, to check how intuitive the notions are, and how usable the current tool is, we selected one person with computer science background, and after making a slide presentation (that took 10 min), and a demo of the tool that took 5 min, we asked her to use two versions of the CIDOC ontology[22] (the first containing 810 schema triples and the second 1,004) and then to migrate and curate a corresponding set of instance triples consisting of 116 instance triples that document the artwork "Avis de Tempete" of Georges Aperghis. From this process, we understood that: (a) the concept and the process is easy to grasp (she understood it from the 10' presentation), (b) in 5 min, she managed to get numerous possible instance triples something that would be impossible without the support of a system.

10.5 Assumptions and Associated Limitations

Single Domain and Range of Properties. In Definition 1 we consider properties which have a single domain and a single range. Note that this is a good practice in conceptual modeling, also in semantic networks, and obligatory in most (or even all) object-oriented systems. We should however note, that RDF/S allows having multiple classes defined as the domain and range of a property. For applying our approach, in such cases, it is suggested to define new classes so that all properties have a single domain (resp. single range) class. In this way, we also avoid some strange inferences. For example, suppose the following three RDF triples (`hasNamedomainDog`), (`hasNamedomain Person`), and (`yannisURIhasNameYannis`). From the above, it is inferred [recall the RDF/S derivation rule (v)] that (`yannisURItypePerson`) and (`yannisURI, typeDog`)! We avoid such cases if properties have single domain and range classes.

[20] For example, there are tools that scan Java bytecode for method calls and create a description of the dependencies between classes and the package/archive encoded in RDF. Other tools transform Maven POM (Project Object Model) files into RDF.

[21] http://leo.hua.gr/palc2012/?q=en.

[22] http://www.cidoc-crm.org/.

The constraint (ii) of Definition 2, that states that the domain and range of a subproperty should be subclasses of (or the same with) the domain and range of its superproperty, is also classical in object-oriented and semantic modeling and makes our propositions and algorithms more clear and elegant.

Acyclicity. Regarding the acyclicity constraint (Definition 2), note that the construction of cyclic schemas is not a good practice in conceptual modeling. In our task, if a non cyclic schema becomes cyclic, then this could result to counterintuitive behavior (from the curation point of view), e.g. too much past negative triples can become certain, etc. However, note that the acyclicity constraint is not used in any of the proofs, therefore this constraint could be eliminated.

11 Related Work

There are several works on ontology evolution and versioning, for a recent overview see [12]. Below we describe in brief some of these works, although they are not directly related with our proposal. At first, we refer to works related to backwards compatible ontology evolution and then to works concerned with data validity and preservation after ontology evolution.

[*On Backwards Compatible Ontology Evolution*]

Klein et al. [15] propose a versioning mechanism for reducing the problems caused by ontology evolution. They argue that ontology versioning is necessary because changes to ontologies may cause incompatibilities, and drive to situations where the new (changed) ontology cannot be used in place of its previous version. They list a number of artifacts that may depend on an ontology, and thus can become incompatible after ontological evolution, and data that conforms to an ontology is one of them. When an ontology is changed, data may get a different interpretation or may use terms that do not exist any more. The authors introduce various forms of compatibility and one of them is *backwards compatibility*. In the same direction, Xuan et al. [31] propose a model to deal with the problem of asynchronous ontology versions in the context of a materialized integration system, which is based on the principle of *ontological continuity*, which refers to the permanence of classes, properties, and subsumption. This principle is actually what we call *backwards compatibility*.

[*Ontology Evolution and Data Validity*]

Noy and Klein [19] present an informal discussion on the differences between ontology evolution and database schema evolution, and how structural changes in ontologies affect the preservation of their data instances. They focus on whether instance data can still be accessed through the changed ontology, and they classify the operation effects as information-preserving changes, translatable changes, and information-loss changes.

Now, Flouris et al. [10,16] focus on the *effects* of a requested change operation, i.e. how the new knowledge base should be after a request for a change, and on its *side-effects* on both schema and instance data, i.e. certain additional actions executed to restore validity. They propose a general-purpose algorithm for determining the effects and side-effects of a requested elementary or complex change operation, and such works can be used to resolve the conflicts.

In addition, Qin and Alturi [22] focus on the validity issue of data instances during ontological evolution. They classify the changes to ontologies into two levels—structural and semantic. Structural changes include changes such as adding/removing classes or properties, modifying the range of a property, etc. Semantic changes mean changes to the semantics presented by the ontology, such as changes to the generalization/descriptiveness of a class or changes to the generalization/restrictiveness of a property. Semantic changes can be further classified into explicit and implicit changes. While explicit semantic changes can be detected based on the structural changes, implicit semantic changes need to be derived from explicit semantic changes by using implication analysis. They propose an algorithm for evaluating the structural validity of a data instance and then another algorithm for evaluating the semantic validity of a data instance.

In our case, since the target schema is fixed, the problem of validity concerns only the instance triples. Mainly there are two approaches to tackle it, and the choice is application specific. According to the first approach (which is actually what we assume in this paper), the set of instance triples should remain the same. In this case if a class c is deleted from S_K we consider that in K', the instances that were explicitly classified in c, i.e. all $(o\ type\ c) \in I_K$, are not affected. Thus, c remains in K' as an unconnected class. Now, if a property pr is deleted from S_K (and also the statements regarding its domain and range are deleted from the initial schema), we consider that in K' the property instance triples $(o\ pr\ o') \in I_K$ are not affected, thus, pr remains in K' as an unconnected property (its domain and range is the top class, i.e. *Resource*).

According to the second approach, we can change the set of instance triples so that K' contains schema elements *only* of $S_{K'}$. Obviously this requires updating I_K and it can be done based on the principles of *minimal change* (reminiscent of the postulates of the AGM theory [1] regarding contraction). For the case of RDF/S this is actually quite straightforward (related work include [16,19]).

In future work, it would be worth to investigate, in the non-backwards compatible schema evolution case, a possible combination of the "repair" of invalid instance descriptions proposed in [10,16] with our proposal for computing possible instance descriptions. That would make sense only

if instead of having as input the entire new schema we receive as input only the changes between the current and the new schema.

In conclusion, there are several works and approaches for dealing with the validity of data during migration, however there is no work for managing their specificity and quality while ontologies evolve.

Finally, we should mention that we do not focus on supporting *historical queries* regarding how concepts evolved over time, as in [24]. In addition, the way we compute possibilities does not presuppose an explicit modeling of how the individual concepts have evolved over time.

At last, [8] proposes a methodology for adding negative constraints to OWL ontologies, i.e. constraints that define expressions that should not be satisfiable. For example, an expression of "vegetarian pizza having meat as ingredient" should not be satisfiable. Instead, in the current paper, we do not focus on satisfiability. We focus on the uncertainty in the context of migrating sets of instance descriptions to new RDF schemas.

12 Concluding Remarks

The rapid evolution of ontologies requires principles, techniques, and tools for managing the quality of the migrated descriptions, as well as flexible interactive methods for managing the incurred uncertainty. To the best of our knowledge this is the first work that exploits ontology schema evolution for managing the specificity of instance descriptions.

In this work, we use *possible instance triples* for expressing (actually bounding) the uncertainty regarding the specificity of a description caused by its migration to a new schema. We introduce the notion of TFP-partition (True-False-Possible partition) of a KB for capturing various application-specific assumptions about the specificity of its descriptions. Subsequently, we formalize the *migration* to a new schema as a *transition* between two TFP-partitions, which should be governed by few (two for the case of backwards compatible, and three for the case of non-backwards compatible schema evolution) *postulates* which are general (i.e. RDF/S independent). Based on these postulates, we derive rules for carrying out a transition for the case of RDF/S. To reduce the space requirements, we show how transitions can be defined without having to keep any negative information (i.e. the "F" part of a TFP-partition), instead only the certain and the possible part of the KB has to be kept, reaching to what we call extended KB (*eKB*). To further compress the possible part of the eKB, and making it feasible for large data sets, we propose a compact (interval-based) representation (exploiting intrinsic properties of TFP-partitions).

Since the ultimate objective is not just the identification of possibilities, but to aid making the instance descriptions as specific as possible, we propose a specificity *lifecycle management process* that *ranks* the possible instance triples, prompts to the user a subset of the possible instance triples and we show how the extended KB should be *updated* when the user *accepts* or *rejects* some of them.

To investigate the feasibility of our approach, we designed and developed a prototype system, and we applied our approach on real and synthetic datasets. The measurements demonstrate the feasibility of the approach.

There are several issues for future research. One interesting direction is to generalize our approach to the *XSD*[23]-typed literal values [21] of property instance triples. Such extension would allow reasoning about the *accuracy* of the migrated descriptions over linearly ordered domains (e.g. as consequence of migrating 32-bit floating numbers to a 64-bit representation).

Acknowledgments Work done in the context of NoE *APARSEN* (Alliance Permanent Access to the Records of Science in Europe, FP7, Proj. No 269977, 2011–2014). Many thanks to Xristina Lantzaki who participated in the user study.

Appendix A: List of symbols

See Table 12.

Appendix B: Example of Applying the Algorithm for Backwards Compatible Migration

Example 9 Consider the example of Fig. 5 and suppose that $P_K = \{$(Fiat_1 type Vehicle), (Bob uses BMW_1), (Alice works at FORTH)$\}$. Executing Algorithm 1, step by step, we have that:

(1) $K' = (S_{K'}, I_K)$;
(2) $P_{K\_Add} = \emptyset$;
(3) $P_{K\_Del} = \emptyset$;
(4) NC = {Van, Jeep, Adult, Institute, University};
(6) $P_{K\_Add} = P_{K\_Add} \cup$ {(Fiat_1 type Van), (BMW_1 type Van), (Fiat_1 type Jeep), (BMW_1 type Jeep), (Bob type Adult), (Alice type Adult), (Computer Science Department type Institute), (FORTH type Institute), (Computer Science Department type *University*), (*FORTH* type University)};
(8) $P_{K\_Del} = P_{K\_Del} \cup \{\emptyset\}$;
(9) NP = {paid from};
(11) $P_{K\_Add} = P_{K\_Add} \cup$ {(Alice paid from Computer Science Department), (Bob paid from FORTH)};
(13) $P_{K\_Del} = P_{K\_Del} \cup$ {(Alice works at FORTH)};

[23] XML Schema Definition.

Table 12 Symbols and description

List of Symbols	
Symbol	Description
$\mathcal{C}(K)$	The *closure* of a KB K
C_K	The set of classes of $\mathcal{C}(K)$
Pr_K	The set of properties of $\mathcal{C}(K)$
$\leq^*_{cl}$	The *subClassOf* relation between classes in C_K
$\leq^*_{pr}$	The *subPropertyOf* relation between properties in Pr_K
Res_K	The resources of a KB K
$Inst_K$	The instances of a KB K
$inst_K(c)$	The instances of a class c of a KB K
B_K	The set of *cartesian instance triples* of a KB K
S_K	The set of *schema triples* of a KB K
I_K	The set of *instance triples* of a KB K
$\mathcal{C}_i(K)$	The instance triples of the closure of a KB K
$valid(o, pr, o', K)$	A *valid* property instance triple of a KB K
$Invalid(K)$	The set of *invalid* property instance triples of a KB K
$SubTriples((o\ type\ c))$	The set of *subtriples* of a class instance triple $(o\ type\ c)$ of a KB K
$SubTriples((o\ pr\ o'))$	The set of *subtriples* of a property instance triple $(o\ pr\ o')$ of a KB K
$SubTriples(A)$	The set of *subtriples* of a set of instance triples A of a KB K
F_K	The set of negative(false) instance triples of a KB K
P_K	The set of possible instance triples of a KB K
P_{K_Add}	The set of added possible instance triples in P_K
P_{K_Del}	The set of deleted possible instance triples from P_K
$posTriples^{cl}(o)$	The set of possible class instance triples of an instance o
$posTriples^{spr}(o)$	The set of possible property instance triples of an instance o as subject
$posTriples^{opr}(o)$	The set of possible property instance triples of an instance o as object
$SuperTriples((o\ type\ c))$	The set of *supertriples* of a class instance triple $(o\ type\ c)$ of a KB K
$SuperTriples((o\ pr\ o'))$	The set of *supertriples* of a property instance triple $(o\ pr\ o')$ of a KB K
$SuperTriples(A)$	The set of *supertriples* of a set of instance triples A of a KB K
K^{up}	The updated certain part of an *eKB*
P^{up}	The updated possible part of an *eKB*
$dist_{cl}(c \rightarrow c')$	The length of the shortest path from a class c to a class c'
$distClass(o, c)$	The shortest distance of c from one of the certain classes of o
$dist_{pr}(pr \rightarrow pr')$	The length of the shortest path from a property pr to a property pr'
$distProperty(o, pr, o')$	The shortest distance of pr from one of the certain properties of (o, o')
$Valid(K)$	The set of *valid* property instance triples of a KB K
$P_K^{\texttt{comp}}$	The set of *composite possibilities*
$P_K^{\texttt{ext}}$	The set of *extended possibilities* including *atomic* and *composite possibilities*
$Cl(o)$	The set of all certain classes of an instance o
$posCl(o)$	The set of all possible classes of an instance o

Note that, (`AlicerelatedtoFORTH`) $\notin (P_K \cup \mathcal{C}_i(K'))$ and `works at` $\leq_{cl}$ `related to` holds in K'. So, we have to move (`Alice works at FORTH`) from P_K to $F_{K'}$ (due to Rule $R2$).

(14) $P_{K\_Del} = P_{K\_Del} \cup$ {(`Fiat_1 type Vehicle`), (`Bob uses BMW_1`)};

Note that P_{K_Del} is updated by those triples that belong to P_K and now belong to $\mathcal{C}_i(K')$. So, we have to remove them from P_K.

(15) $P_{K'} = P_K \setminus P_{K\_Del} =$
{(`Fiat_1 type Vehicle`), (`Bob uses BMW_1`), (`Alice works at FORTH`)}\

{(Fiat_1 type Vehicle), (Bob uses BMW_1), (Alice works at FORTH)} = ∅;

(16) $P_{K'}$ = {(Fiat_1 type Van), (BMW_1 type Van), (Fiat_1 type Jeep), (BMW_1 type Jeep), (Bob type Adult), (Alice type Adult), (Computer Science Department type Institute), (FORTH type Institute), (Computer Science Department type University), (FORTH type University), (Alice paid from Computer Science Department), (Bob paid from FORTH)}

(17) Return $P_{K'}$;

In order to explain line 8 in part B of Algorithm 1, consider Fig. 5. Suppose that we have another version $S_{K''} = S_{K'} \cup \{(\texttt{Van} \leq_{cl} \texttt{LoadCarrying Vehicle})\}$, where we have a new specialization relationship, i.e. Van $\leq_{cl}$ LoadCarrying Vehicle. Then, according to line 8 of Algorithm 1, we have that $P_{K\_Del} = P_{K\_Del} \cup \{(\texttt{Fiat\_1 type Van})\}$. This is because it holds that (Fiat_1 type LoadCarrying Vehicle) $\notin P_{K'} \cup \mathcal{C}_i(K'')$ and (Van $\leq_{cl}$ LoadCarrying Vehicle) $\in S_{K''}$. So, we have to move (Fiat_1 type Van) from $P_{K'}$ to $F_{K''}$ (due to Rule $R1$).

Note that if $(o\ type\ c) \in \mathcal{C}_i(K')$ and $(o\ type\ c') \notin \mathcal{C}_i(K') \cup P_{K'}$, for all $c' \leq^*_{cl} c$ and $c' \neq c$, then the *MSA* property holds for the class instance triple $(o\ type\ c)$. Similarly, if $(o\ pr\ o') \in \mathcal{C}_i(K')$ and $(o\ pr'\ o') \notin \mathcal{C}_i(K') \cup P_{K'}$, for all $pr' \leq^*_{pr} pr$ and $pr' \neq pr$, then the *MSA* property holds for the property instance triple $(o\ pr\ o')$.

References

1. Alchourrón CE, Gärdenfors P, Makinson D (1985) On the logic of theory change: partial meet contraction and revision functions. J Symb Logic 50(2):510–530
2. Bizer C, Heath T, Berners-Lee T (2009) Linked data-the story so far. Int J Semant Web Inform Syst 5(3):1–22
3. Brickley D, Guha RV (2004) RDF Vocabulary Description Language 1.0: RDF Schema, W3C Recommendation, February 2004. http://www.w3.org/TR/rdf-schema/
4. Buneman P, Cheney J, Wang Chiew T, Vansummeren S (2008) Curated Databases. In 27th ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems (PODS-2008), pp 1–12
5. Christophides V, Plexousakis D, Scholl M, Tourtounis S (2003) On labeling schemes for the semantic web. In: 12th International world wide web conference (WWW-2003), pp 544–555
6. Dalal M (1988) Investigations into a theory of knowledge base revision: preliminary report. In: 7th National conference on artificial intelligence (AAAI-1988), pp 475–479
7. Fernández JD, Martínez-Prieto MA, Gutierrez C (2010) Compact representation of large RDF data sets for publishing and exchange. In: 9th international semantic web conference (ISWC-2010), pp 193–208
8. Ferré S, Rudolph S (2012) Advocatus diaboli-exploratory enrichment of ontologies with negative constraints. In: 18th international conference on knowledge engineering and, knowledge management (EKAW-2012), pp 42–56
9. Flouris G (2006) On belief change and ontology evolution. PhD thesis, Computer Science Department, University of Crete, Greece
10. Flouris G, Konstantinidis G, Antoniou G, Christophides V (2013) Formal foundations for RDF/S KB evolution. Int J Knowl Inform Syst (KAIS) 35(1):153–191
11. Gutierrez C, Hurtado C, Mendelzon A (2004) Foundations of semantic web databases. In: 23rd ACM symposium on principles of database systems (PODS-2004), pp 95–106
12. Hartung M, Terwilliger J, Rahm E (2011) Recent advances in schema and ontology evolution. Schema matching and mapping
13. Hayes P (2004) RDF semantics, W3C recommendation, February 2004. http://www.w3.org/TR/rdf-mt/
14. Jin R, Xiang Y, Ruan N, Wang H (2008) Efficiently answering reachability queries on very large directed graphs. In: ACM SIGMOD international conference on management of data (SIGMOD-2008), pp 595–608
15. Klein MCA, Fensel D (2001) Ontology versioning on the semantic web. In: First semantic web working symposium (SWWS-2001), pp 75–91
16. Konstantinidis G, Flouris, G, Antoniou G, Christophides V (2008) A formal approach for RDF/S ontology evolution. In: 18th European conference on artificial intelligence (ECAI-2008), pp 70–74. Patras, Greece, July 2008
17. Neumann T, Weikum G (2008) RDF-3X: a RISC-style engine for RDF. Proc VLDB Endow (PVLDB) 1(1):647–659
18. Neumann T, Weikum G (2010) x-RDF-3X: fast querying, high update rates, and consistency for RDF databases. Proc VLDB Endow (PVLDB) 3(1):256–263
19. Noy NF, Klein MCA (2004) Ontology evolution: not the same as schema evolution. Knowl Inform Syst 6(4):428–440
20. APARSEN: Network of Excellence (FP7 No 269977) 2011–2014. Deliverable D26.1: Report and Strategy on Annotation, Reputation and Data Quality, 2012. http://www.alliancepermanentaccess.org/wp-content/uploads/downloads/2012/12/APARSEN-REP-D26_1-01-1_0.pdf
21. Peterson D, (Sandy) Gao S, Malhotra A, Sperberg-McQueen, CM, Thompson HS (2009) W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes, W3C Working Draft 3, December 2009. http://www.w3.org/TR/xmlschema11-2/
22. Qin L, Atluri V (2009) Evaluating the validity of data instances against ontology evolution over the Semantic Web. Inform Softw Technol 51(1):83–97
23. Rada R, Mili H, Bicknell E, Blettner M (1989) Development and application of a metric on semantic nets. IEEE Trans Syst Man Cybern 19(1):17–30
24. Rizzolo F, Velegrakis Y, Mylopoulos J, Bykau S (2009) Modeling concept evolution: a historical perspective. In: 28th International conference on conceptual modeling (ER-2009), pp 331–345
25. Sahoo SS, Halb W, Hellmann S, Idehen K, Thibodeau T Jr, Auer S, Sequeda J, Ezzat A (2009) A Survey of current approaches for mapping of relational databases to RDF. Report by the W3C RDB2RDF incubator group. http://www.w3.org/2005/Incubator/rdb2rdf/RDB2RDF_SurveyReport.pdf
26. Theodoridou M, Tzitzikas Y, Doerr M, Marketakis Y, Melessanakis V (2010) Modeling and querying provenance by extending CIDOC CRM. Distrib Parallel Databases 27(2):169–210
27. Theoharis Y, Georgakopoulos G, Christophides V (2007) On the synthetic generation of semantic web schemas. In: Joint ODBIS & SWDB workshop on semantic web, ontologies, and databases. Collocated with VLDB2007, pp 98–116

28. Theoharis Y, Tzitzikas Y, Kotzinos D, Christophides V (2008) On graph features of semantic web schemas. IEEE Trans Knowl Data Eng 20(5):692–702
29. Tzitzikas Y, Kampouraki M, Analyti A (2012) Curating the specificity of metadata while World Models Evolve. In: 9th annual iPres conference on digital preservation (iPres-2012). http://www.ics.forth.gr/analyti/Local_Papers/CuratingSpecificity_iPres_CRC_pv.pdf
30. Wang H, He H, Yang J, Yu PS, Yu JX (2006) Dual labeling: answering graph reachability queries in constant time. In: 22nd international conference on data engineering (ICDE-2006)
31. Xuan DN, Bellatreche L, Pierra G (2006) A versioning management model for ontology-based data warehouses. In: 8th International conference on data warehousing and knowledge discovery (DaWaK 2006), pp 195–206
32. Zeginis D, Tzitzikas Y, Christophides V (2007) On the foundations of computing deltas between RDF models. In: 6th international semantic web conference (ISWC-07), pp 637–651. Springer, Berlin
33. Zeginis D, Tzitzikas Y, Christophides V (2011) On computing deltas of RDF/S knowledge bases. ACM Trans Web (TWEB) 5(3):14