Information Systems 42 (2014) 1–14

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

Target-driven merging of taxonomies with Atom

Salvatore Raunich *, Erhard Rahm

University of Leipzig, Germany

ARTICLE INFO

Article history:
Received 27 May 2013
Accepted 12 November 2013
Recommended by: J. Van den Bussche
Available online 25 November 2013

Keywords:
Data models
Data integration
Taxonomy merging
Database applications

ABSTRACT

The proliferation of ontologies and taxonomies in many domains increasingly demands the integration of multiple such ontologies. We propose a new taxonomy merging algorithm called Atom that, given as input two taxonomies and a match mapping between them, can generate an integrated taxonomy in a largely automatic manner. The approach is target-driven, i.e. we merge a source taxonomy into the target taxonomy and preserve the target ontology as much as possible. In contrast to previous approaches, Atom does not aim at fully preserving all input concepts and relationships but strives to reduce the semantic heterogeneity of the merge results for improved understandability. Atom can also exploit advanced match mappings containing is-a relationships in addition to equivalence relationships between concepts of the input taxonomies. We evaluate Atom for synthetic and real-world scenarios and compare it with a full merge solution.

Euler has all RCC5 relations (including disjunctions over these, to represent incomplete information)

1. Introduction

Ontologies and taxonomies are increasingly used to semantically categorize or annotate information, especially on the web. For example, product catalogs of online shops or web directories categorize products or websites to help users finding relevant entries. In life sciences, ontologies are used to describe components and functions of organisms or objects such as genes or proteins. Since many ontologies refer to the same domain and to the same objects, there is a growing need to integrate or merge such related ontologies. The goal is to create a merged ontology providing a unified view on the two or more input ontologies.

Despite a significant amount of previous work on the related problem of schema integration [2], ontology integration is still a challenge and not sufficiently solved. Previous ontology merging approaches [17,13,32] are largely user-controlled and provide little support to automatically determine merge solutions. However, such manual approaches are insufficient for merging large ontologies with thousands of concepts so that there is a strong need to automatically determine ontology merge results which the user can confirm or adjust as needed. One promising approach to this end is to decompose the complex integration problem into match and merge subtasks and leverage the significant advances already made for automatic ontology matching to solve the first subproblem. The merge subtask can then utilize a match mapping identifying corresponding concepts in the input ontologies that should be merged. This idea has already been applied for integrating database schemas, where several proposed approaches merge schemas based on a pre-determined match mapping [6,28,20,29,23].

Previous merge approaches commonly treat all input ontologies symmetrically and require that all information from the input ontologies should be preserved in the merged ontology, in particular all concepts and their relationships [22]. We argue that such symmetric, fully information-preserving merge solutions are not always desirable but may introduce a significant amount of semantic redundancy due to heterogeneous organizations of the same concepts.

For illustration, consider the simple scenario in Fig. 1 that we will use as a running example. The task is to merge

* Corresponding author.
E-mail addresses: raunich@informatik.uni-leipzig.de (S. Raunich), rahm@informatik.uni-leipzig.de (E. Rahm).

0306-4379/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.is.2013.11.001

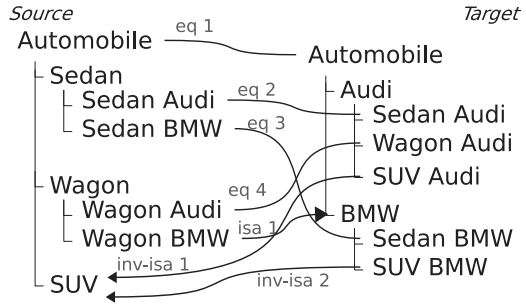

Fig. 1. Running example.

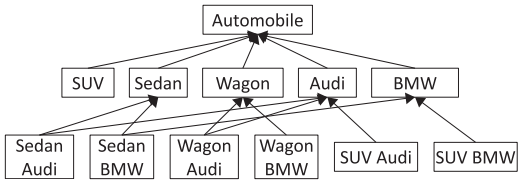

Fig. 2. Full merge solution.

the catalog of a new online car shop (source) into the catalog of a price comparison portal (target). We assume that a match mapping, expressed as a set of correspondences between source and target concepts, is already given, either automatically generated by a matching tool or manually designed by an expert user. In this example, the input matching contains four equivalence correspondences labeled eq_1, eq_2, eq_3, eq_4 (we initially ignore the other correspondences). A typical merge approach would combine equivalent concepts and maintain all the remaining input concepts and relationships in the merge result. We call this a symmetric, *full merge* approach since it preserves all input concepts and relationships.

The running example shows that the two taxonomies organize the vehicles in different ways. The target initially categorizes first by manufacturer (Audi, BMW, etc.) and then by body style (sedan, wagon, etc.) while the source taxonomy uses the opposite order. Fig. 2 shows the solution that a full merge approach would produce. It preserves both views in the merged taxonomy but thereby introduces a semantic overlap (redundancy) and reduces the understandability of the resulting taxonomy. In particular, multiple inheritance has been introduced for several concepts so that there are multiple paths to several leaves. For example, the leaf concept *Sedan Audi* can be reached through both the concepts *Sedan* and *Audi* showing a semantic overlap between these two concepts.

In our new, *asymmetric merge* approach we will deal with such situations by giving preference to the target taxonomy. We merge the source taxonomy into the target taxonomy and only preserve the concepts and structure of the target taxonomy but drop concepts and relationships from the source taxonomy that would introduce redundancy in the merge result. We believe that such an asymmetric merge is highly relevant in practice. It supports the incremental integration of new source ontologies into an existing target ontology, such as a data warehouse or a mediator ontology. Preserving the target ontology can greatly improve its stability and minimize the need to change applications of the integrated ontology. The asymmetric merge is also useful for applications such as web data integration or the integration of life science ontologies. As in the running example, it supports catalog integration of web shops, e.g. for adding the catalog of a new online shop into the catalog of a price comparison portal. In life sciences, there exist already large manually curated hub ontologies such as Uberon or UMLS combining diverse anatomy or other biomedical ontologies [15,5]. Adding further ontologies to such integrated ontologies can benefit from an automatic, asymmetric merge that reduces human effort and leaves the existing target ontologies largely stable.

In particular, we make the following contributions:

- We propose a largely automatic approach for taxonomy merging called Atom which utilizes a given match mapping between the input taxonomies. Atom is an asymmetric, target-driven algorithm, i.e., it merges a source taxonomy into the target taxonomy.
- We propose to restrict the semantic overlap in the merge result for improved understandability. This is achieved by giving preference to the target taxonomy when the same concepts are differently organized in the input taxonomies and limiting the degree of multiple inheritance.
- We propose the use of extended match mappings containing equivalence, is-a and inverse is-a relationships between concepts of the input taxonomies. The additional types of correspondences are used for a better placement of source concepts and to further reduce the semantic overlap in the merge result.
- We have implemented Atom and a full merge solution in a working prototype [26] and present an evaluation of both approaches for medium and large real-life ontologies.

In the next section, we introduce our ontology model and define the main requirements for taxonomy merging. In Section 3, we describe the Atom merge algorithm in detail and discuss its complexity. Section 4 sketches the generation of mappings between the input taxonomies and the merge result that can be used for instance migration. In Section 5 we evaluate the algorithms on real-life ontologies. Related work is described in Section 6 before we conclude.

2. Models and problem definition

2.1. Preliminaries

We first define data representation models used in the paper. An ontology is a quadruple $O = (C, C_i, I, R)$ where C is a collection of *Classes* or *Concepts*, $C_i \subseteq C$ is the subset of concepts containing instances, I is the set of instances associated to classes in Ci possibly empty, and R is the set of *relationships* between concepts. Each concept C has a name (or *label*) and a collection of attributes or properties A_c, possibly empty. Several kinds of relationships can be defined, like “is-a” or “subclass”, “part-of”, “type-of”, etc. A relationship $r(a, b) \in R$ is a directed, binary and semantic

connection between two concepts a and b. It can be explicitly present in the ontology or *implied* by an ontology rule. For example, given two is-a relationships $r(a,b)$ and $r(b,c)$, the relationship $r(a,c)$ is implied since is-a relationships are transitive.

Graphically, as we can see in Fig. 1, we represent concepts with a simple label and, in case of single inheritance, we use a nesting notation to represent is-a relationships. For example, in the source ontology, the concept *Wagon Audi* is a subclass of the concept *Wagon*. It is important to note that, in general, a taxonomy is a graph but, in this example, we use a tree-style representation to simplify the visualization.

In this paper, we will consider only ontologies $O=(C, C_i, I, R)$ where C_i contains only leaf nodes and R contains only "is-a" relationships between concepts. For this reason, in the following, we will use the terms *ontology* and *taxonomy* with the same meaning. Our taxonomies are acyclic but multiple inheritance is supported, i.e. a concept can have multiple parents.

The (match) mapping between two ontologies $O_S=(C_s, C_{is}, I_s, R_s)$ and $O_T=(C_t, C_{it}, I_t, R_t)$ is defined as a set of *correspondences*. Correspondences are either *concept correspondences* or *attribute correspondences*. Given two concepts $s \in C_s$ and $t \in C_t$, we define a concept correspondence (s,t) as an ordered pair of a source concept and a target concept. Similarly, given two attributes $a_s \in A_s$ and $a_t \in A_t$, we define an attribute correspondence (a_s, a_t) as an ordered pair of a source attribute and a target attribute. Each concept correspondence is characterized by a type selected from *equivalence*, *is-a* and *inverse-isa*. Semantically, we define an equivalence correspondence (s,t) if s and t represent the same concept; similarly, a correspondence (s,t) is an is-a correspondence if s is a subclass of t while (s,t) is an inverse-isa correspondence if t is a subclass of s. While inverse-isa relationships are the dual of is-a relationships we need to distinguish because correspondences are ordered in our model. Furthermore, our algorithm is target-driven and thus requires a different treatment for is-a and inverse is-a relationships. Examples of concept correspondences are shown in Fig. 1. In the following, we refer to mappings containing not only equivalence correspondences but also semantic is-a and inverse-isa correspondences as *extended mappings*.

In this paper, we only consider 1:1 equivalence mappings. However, we can deal with mutliple is-a or inverse is-a correspondences referring to the same source or target concept, e.g., as for source concept *SUV* in Fig. 1. We do not investigate here how these correspondences are generated but assume that a *correct* and *complete* match mapping is provided, automatically generated by a matching tool or manually determined by a domain expert.

2.2. Problem definition and properties of the merge result

Given the input taxonomies $O_S=(C_s, C_{is}, I_s, R_s)$ and $O_T=(C_t, C_{it}, I_t, R_t)$ and the input match mapping between O_S and O_T, Map_{ST}, the goal is to automatically determine the merged taxonomy $O_{T'}=(C_{t'}, C_{it'}, I_{t'}, R_{t'})$ as well as output mappings $Map_{ST'}$ between O_S and $O_{T'}$ and $Map_{TT'}$ between O_T and $O_{T'}$, such that a set of properties hold.

We distinguish two sets of properties (requirements) to differentiate between a full merge solution and the Atom approach. Based on requirements for a generic schema merge approach introduced in [22], we first introduce four properties for a merge that fully preserves both input taxonomies. For the asymmetric merge approach we will then relax the first two properties but add three additional ones.

(P1′) *Element preservation*: Each element (a concept or an attribute) in the input taxonomies O_S and O_T has a corresponding element in the merge result $O_{T'}$. Formally, each concept $c \in C_s \cup C_t$ corresponds to exactly one concept $c' \in C_{t'}$. This equivalence concept correspondence is defined as $(c,c') \in Map_{ST'} \cup Map_{TT'}$. Similarly, each attribute $a \in A_c$ corresponds to exactly one attribute $a' \in A_{c'}$. This attribute correspondence is defined as $(a,a') \in Map_{ST'} \cup Map_{TT'}$.

(P2′) *Relationship preservation*: Each input is-a relationship is explicitly in or implied by $O_{T'}$. Formally, for each is-a relationship $r(s,t) \in R_s \cup R_t$, if $(s,s') \in Map_{ST'} \cup Map_{TT'}$ and $(t,t') \in Map_{ST'} \cup Map_{TT'}$, then either $r(s',t') \in R_{t'}$ or $r(s',t')$ is implied in $O_{T'}$.

(P3) *Instance preservation*: All instances of both input taxonomies must be preserved in the merged taxonomy. Moreover, no instance overlap between concepts should be introduced, i.e. every instance should migrate to exactly one concept in the merge result. Formally, for each source or target concept containing instances $c_i \in C_{is} \cup C_{it}$ there should be at least one correspondence $(c_i, c_i') \in Map_{ST'} \cup Map_{TT'}$ to an instance-containing concept $c_i' \in C_{it'}$ in the merge result. Furthermore, each instance i of c_i should be uniquely migrated by a transformation function f to an instance $i'=f(i)$, $i' \in I_{t'}$ of an instance-containing concept in $C_{it'}$.

Under our assumption that instances are restricted to leaf concepts we can ensure property P3 by preserving all leaf nodes of both the source and target taxonomies. Preserving the leaves is generally desirable even when there are no instances since leaf concepts represent the most specific knowledge in a taxonomy.

(P4) *Equality preservation*: If two concepts s and t are equal in Map_{ST} then they are mapped to the same merged concept in the result $O_{T'}$ and vice versa. Formally, if two concepts $s,t \in C_s \cup C_t$ are equal in Map_{ST}, then there exists a unique concept $c \in C_{t'}$ such that $(s,c) \in Map_{ST'}$ and $(t,c) \in Map_{TT'}$. If s and t are not equal in Map_{ST}, then such a concept c does not exist and s and t correspond to different elements in $O_{T'}$.

It is easy to see that the full merge solution shown in Fig. 2 satisfies these properties. We now turn to the requirements for a target-driven merge solution that aims at reducing the semantic overlap and redundancy that might be created by a fully information preserving approach. For such an approach, we keep properties (P3) and (P4) but only require the preservation of the target taxonomy. We thus replace properties (P1′) and (P2′) by the following requirements:

(P1) *Target element preservation*: Each concept $c \in C_t$ is preserved so that there is exactly one corresponding

concept $c' \in C_{t'}$. This concept correspondence is defined as $(c, c') \in Map_{TT'}$. Similarly, each attribute $a \in A_c$ corresponds to exactly one attribute $a' \in A_{c'}$. This attribute correspondence is defined as $(a, a') \in Map_{TT'}$.

(P2) *Target relationship preservation*: For each is-a target relationship $r(s,t) \in R_t$, if $(s,s') \in Map_{TT'}$ and $(t,t') \in Map_{TT'}$, then either $r(s',t') \in R_{t'}$ or $r(s',t')$ is implied in $O_{T'}$.

Moreover, we add a new requirement P5 in order to limit the redundancy in the merge result.

(P5) *Control of semantic overlap*: The merge algorithm should generate an integrated taxonomy with little or no redundancy compared to the input taxonomies. In particular, we want to avoid or limit multiple paths to leaf nodes introduced by different concept organizations in the input taxonomies. We thus require that for a leaf concept of the target taxonomy no redundant paths are introduced by the merge. Formally, for each pair of concepts $c \in C_t$ and $c' \in C_{t'}$ such that c is a leaf concept and there exists $(c, c') \in Map_{TT'}$, the number of different paths to c in O_T and c' in $O_{T'}$ should be the same.

We remark that if O_T is a tree-structured taxonomy, i.e. O_T has no multiple inheritance, then (P5) implies that the merge result $O_{T'}$ will remain a tree and no multiple inheritance will be introduced. This property is obviously not satisfied by the full merge result shown in Fig. 2.

Finally, since we can also provide is-a and inverse-isa relationships in Map_{ST}, we add the following requirements P6 and P7 to define their impact in the result $O_{T'}$.

(P6) *Is-a correspondences preservation*: If two concepts are defined so that the first is a subclass of the second one in Map_{ST} then a similar is-a relationship is explicitly in or implied by $O_{T'}$. Formally, given two concepts $s, t \in C_s \cup C_t$, if there exists an is-a correspondence $(s,t) \in Map_{ST}$ and if $(s,s') \in Map_{ST'}$ and $(t,t') \in Map_{TT'}$, then either $r(s',t') \in R_{t'}$ or $r(s',t')$ is implied in $O_{T'}$.

(P7) *Inverse is-a correspondences preservation*: If two concepts are defined so that the second is a subclass of the first one in Map_{ST} then either a similar relationship is explicitly in or implied by $O_{T'}$ or the two concepts are combined in $O_{T'}$. Formally, given two concepts $s, t \in C_s \cup C_t$, if there exists an inverse is-a correspondence $(s,t) \in Map_{ST}$ and if $(s,s') \in Map_{ST'}$ and $(t,t') \in Map_{TT'}$, then either $r(t',s') \in R_{t'}$ or $r(t',s')$ is implied in $O_{T'}$ or $s' = t'$.

Property P7 suggests two possible ways to translate inverse is-a correspondences, generating a new is-a relationship between the corresponding concepts or combining them. While the first approach adds a new parent to the target concept slightly changing the target structure (but still satisfying all the requirements above), the second approach simply combines input concepts keeping the solution more compact.

As a general requirement, the merge algorithm must terminate and produce a result that is itself a taxonomy (respectively *termination* and *closure*); we also ask for good *efficiency* and *scalability* providing acceptable execution times even for large taxonomies with many concepts and is-a relationships. As the distinction between the two possible merge solutions shows, there are generally different valid merge solutions so that a user should be able to interactively influence the merge result. We will not focus on possible strategies for such user interaction but describe the largely automatic generation of a default solution satisfying the introduced properties and assume that user interaction is performed in a post-processing step.

3. The merge algorithm

We propose a merge algorithm that, given as input two taxonomies and a set of correspondences, produces an integrated taxonomy that meets the requirements introduced in Section 2. We split the description of the algorithm in to two successive phases: a *preliminary phase* that, starting from the source taxonomies and the set of correspondences, generates a so-called integrated concept graph; and a *main phase* that uses the integrated concept graph to generate the final result. We focus on generating the merged taxonomy in this section; the generation of the output mappings is explained in Section 4.

3.1. Preliminary phase

The preliminary phase takes as input two taxonomies O_S and O_T and a match mapping between them, provided as a set of concept correspondences and attribute correspondences, and its goal is the generation of an integrated concept graph I. A similar algorithm was introduced in [6] for relational and XML schemas and the preliminary phase of our approach is based on it but with significant differences that we will discuss later in this section. The pseudo-code is described in Algorithm 1 in the Appendix.

The input taxonomies O_S and O_T are represented as two directed concept graphs G_S and G_T, respectively, where nodes are concepts and edges are is-a relationships.

To generate an integrated concept graph I, we first generate an integrated concept C for each pair of concepts that have an input equivalence correspondence at the concept or attribute level. The assumption of 1:1 equivalence correspondences ensures that only pairs of concepts are merged in this way.

For an integrated concept C we derive a unique label l from the merged concepts. Let l_s and l_t be the labels of the merged concepts s and t, we define the label of the integrated concept C as $label(C) = genLabel(l_s, l_t)$. The function $genLabel()$ can be defined in different ways. For ease and clarity of exposition we chose to simply concatenate the labels of the concepts to be integrated, and if $l_s = l_t$ we use their common string value once and append a "*" symbol. For example, the label for the integrated concept that merges the source concept *Sedan Audi* with the target concept *Sedan Audi* is *label*= *Sedan Audi**. Note, that this labeling convention is only used internally for the algorithm and that the real labels of merged concepts should be more meaningful, e.g. by keeping the labels of the target concepts.

For two attributes involved in an attribute correspondence, we add only one attribute with a label generated by a similar function as for concept labels.

At this point, for each integrated concept C we generate a new node in I. Then, we translate the set of edges in G_S and G_T into a new set of labeled edges in I. In particular, for each edge $e = C_1 \rightarrow C_2$ in G_S, we produce an *S-labeled*

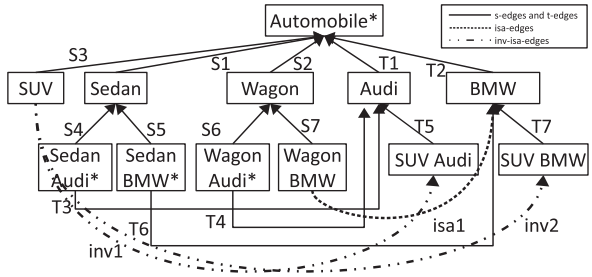

Fig. 3. Integrated concept graph for running example.

edge in I – in the following called *s-edge* – defined as $e_s = C_{i1} \rightarrow C_{i2}$ where C_{i1} and C_{i2} are the corresponding integrated concepts in I of C_1 and C_2. Similarly, for each edge e in G_T, we produce a *T-labeled edge* (or *t-edge*) in I.

Finally, for each is-a and inverse-isa correspondence we generate an *isa-edge* and an *inv-isa-edge*, respectively, in I. The integrated concept graph built for our running example is in Fig. 3. Source edges and target edges are represented by solid lines, isa-edges and inv-isa-edges by dashed and dotted lines, respectively. The graph contains four integrated concepts (labels with $*$), s-edges $S1$–$S7$, t-edges $T1$–$T7$, one isa-edge and two inv-isa-edges.

The integrated concept graph (without isa- and inv-isa edges) can be used to derive the full merge result in a straightforward manner, as can be seen by the high similarity of Fig. 3 with Fig. 2. The Atom approach differs from such an approach by its focus on preserving the target taxonomy, reducing semantic overlap and utilizing extended input correspondences. These aspects lead already to several changes in the preliminary phase compared to [6], such as the distinction in the integrated concept graph between source edges and target edges and support for extended (match) mappings.

3.2. Main phase

The main phase of the Atom approach implements its key features already mentioned in the Introduction in order to achieve properties P1–P7 introduced in Section 2. Based on the integrated concept graph we first take over the target concepts and relationships in the merge result. To preserve all instances, we include all leaf nodes from both the target and source taxonomies into the merge result. To reduce semantic overlap, we only consider inner concepts of the source taxonomy if they are considered "relevant", i.e. they do not introduce redundant paths to leaf nodes. We also incorporate is-a and inverse-isa relationships for improving the merge result. Before describing the details of the algorithm, we need to introduce some definitions.

A *path* P in a graph G is a sequence of nodes such that, for each node there exists an edge between this node and the next one in the sequence. A path has a *start node* and an *end node*; the other nodes are called *internal nodes*.

Definition (*Top Level Concepts*). A concept C of the integrated concept graph G is a *Source Top Level Concept* if C has no outgoing s-edges but at least one incoming s-edge. This is equivalent to say that C has one or more children but no parent with respect to s-edges. Similarly, C is a *Target Top Level Concept* if it has no outgoing t-edges in G but at least one incoming t-edge. Finally C is a *Top Level Concept* if it is either a Source or a Target Top Level Concept.

In the graph shown in Fig. 3, the concept *Automobile** is both a source and a target top level concept since it has only incoming s-edges and t-edges but no outgoing edges. As we will discuss later in this section, *Automobile** can be set as root of I since there are no other top level concepts in I.

A path P is called a *TLC path* if its end node is a Top Level Concept in the graph. A path P in a graph G is a *cycle* if the start node and the end node are the same. The choice of start and end nodes is arbitrary.

Let N be a node in a graph G and P be a TLC path with start node N. P is a *source-path* or simply *s-path* if it contains only s-edges. Similarly, P is a *target-path* or simply *t-path* if it contains only t-edges. We define P as a *combined-path* or *c-path* if it contains only s-edges or isa-edges. In Fig. 3, for example, $P_1 = \{S_4\text{–}S_1\}$ is a s-path and $P_2 = \{T_4\text{–}T_1\}$ a t-path.

Now we are ready to discuss in detail the main phase of the algorithm. Given an integrated concept graph I, the algorithm generates an integrated taxonomy $O_{T'}$ using the following steps:

- *Step* 2.1: Removing cycles in I.
- *Step* 2.2: Translation of t-edges.
- *Step* 2.3: Translation of isa-edges.
- *Step* 2.4: Translation of s-edges.
- *Step* 2.5: Translation of inv-isa-edges.
- *Step* 2.6: Finding root concept.

Steps 2.3 and 2.5 are only needed for the extended mapping while with equivalence mappings a simpler algorithm is sufficient.

In the following we detail each step of the algorithm and we show the intermediate and final results for our running example. The pseudo-code is shown in Algorithm 2 in the Appendix.

Step 2.1 [*removing cycles*]: In this step we check whether cycles are present in the graph I in order to remove them. If we assume correct, acyclic source and target taxonomies then the source and the target concept graphs G_S and G_T are cycle-free so that any cycle in I cannot involve only s-edges or t-edges. If we further assume correct correspondences, then all concepts involved in a cycle could be considered equal due to the transitivity of is-a relations. Pottinger and Bernstein [22] thus proposed to merge together all concepts involved in an is-a cycle. We follow a more conservative approach by preserving the target taxonomy (here, its part involved in the cycle) and breaking a cycle by removing one of the involved s-edges. In general, there can be multiple such s-edges and thus different alternatives to resolve a cycle with potentially different merge results. While we provide a default approach to automatically select a cycle-breaking s-edge we recommend to interactively request user feedback to resolve a cycle. This enables the user to verify whether the cycle is

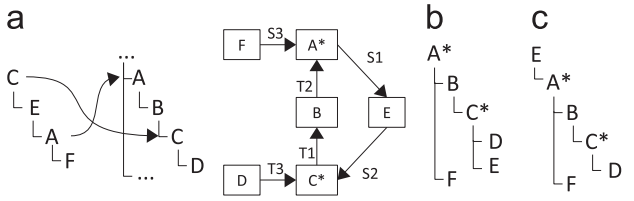

Fig. 4. (a) Example of a cycle, (b) and (c) possible results after removing cycles.

introduced by an error in the match mapping or in the input taxonomies. Furthermore, the user can select which s-edge should be removed to break the cycle.

To give a simple example, the integrated concept graph in Fig. 4(a) has a cycle on the set of nodes {A^*, E, C^*, B} and on the set of edges {S_1, S_2, T_1, T_2}. In this example, the cycle can be broken in to two different ways depending on which s-edge will be removed. Fig. 4(b) and (c) shows the two possible results: the solution in Fig. 4(b) is obtained by removing the s-edge S_1 while Fig. 4(c) shows the result generated by removing S_2. Since the results are semantically different, we leave the final decision to the user.

Step 2.2 [*translation of t-edges*]: For each t-edge $e = N_1 \rightarrow N_2$ we normally create an is-a relationship between the corresponding concepts C_1 and C_2 in $O_{T'}$, in order to maintain the target concepts and relationships. The only exception is when there exists exactly one c-path P with start node N_1 and end node N_2 containing more than one s-edge or isa-edge. In the latter case we do not create a direct relationship between C_1 and C_2 but we mark all edges in P as relevant (for the merged taxonomy) so that they will be translated in the next step. The reason is that we want to preserve the target structure in the final result but if two concepts have a more detailed structure in the source, we want to reward it in the merged taxonomy since it preserves and extends the target structuring between N_1 and N_2, and this is possible thanks to the transitivity of is-a relationships.

Step 2.3 [*translation of isa-edges*]: The intuition behind the translation of an isa-edge is that it represents a subclass relationship between a source concept and a target concept and we want this relationship to be translated also in the integrated taxonomy. For each isa-edge $e = N_1 \rightarrow N_2$, we create an is-a relationship between the corresponding concepts C_1 and C_2. If we consider the graph shown in Fig. 3, we translate the single isa-edge with label "isa_1" nesting the concept *Wagon BMW* in *BMW* (see merge result in Fig. 5(a)).

Step 2.4 [*translation of s-edges*]: The translation of s-edges is the most important step in the algorithm, because it integrates missing source concepts into $O_{T'}$, in a "correct" position, and translates only that concepts that do not introduce redundancy in the solution. First of all, we define *candidates* as the set of all leaf nodes L such that L has at least one outgoing s-edge. These source leaf concepts will be integrated into $O_{T'}$ as well as inner source concepts that do not introduce redundant paths to the source leaf concepts. For each node L we find all s-paths with start node L and for each s-path P we check which of its edges are relevant for the merge results without introducing redundancy in addition to the target edges

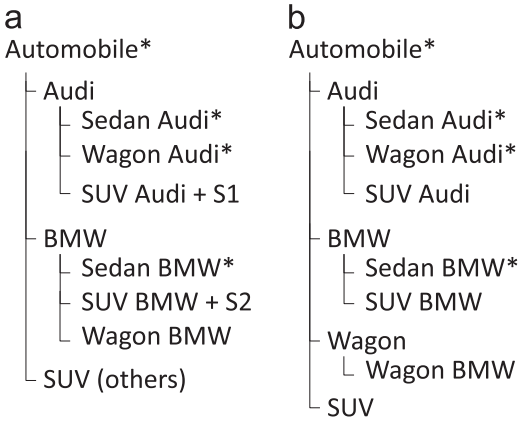

Fig. 5. Result with (a) and without (b) extended mapping.

that will be translated. We therefore traverse each s-path P and consider its edges as relevant until reaching a node in P with outgoing t-edges or isa-edges indicating that the remaining path is already covered by $O_{T'}$. This criterion indicates that such edges have already been identified as relevant in the previous steps so that we do not have to translate them again. At this point we can also decide which source nodes should be integrated in the final result and which not. We define a node X as *not relevant* if all outgoing and incoming s-edges are not marked as relevant. For each s-edge $e = N_1 \rightarrow N_2$ marked as relevant, we create an is-a relationship between the corresponding concepts C_1 and C_2 in $O_{T'}$.

If we apply this step to our running example shown in Fig. 3, the set of candidate nodes is *candidates* = {*SUV*, *Sedan Audi**, *Sedan BMW**, *Wagon Audi**, *Wagon BMW*}. Since only the s-path with start node *SUV* has no concepts with outgoing t-edges or isa-edges, we mark the edge S_3 as relevant and the remaining s-edges (S_1, S_2, S_4, S_5, S_6, S_7) as not relevant. As a consequence, the nodes *Sedan* and *Wagon* are considered not relevant and they will not be translated in the integrated taxonomy.

Step 2.5 [*translation of inv-isa edges*]: An inverse-isa mapping semantically describes how a source concept can be split into two or more target concepts, i.e. there are normally several inverse-isa correspondences for the same source concept to different target concepts. Such 1:n relationships require special treatment to correctly migrate (partition) source instances among several target concepts. For example, in Fig. 1 the source concept *SUV* represents all kinds of SUV without distinction of manufacturer, and it should be split into two target concepts *SUV Audi* and *SUV BMW*. But, in general, *SUV* can also contain vehicles with a manufacturer that is different from both Audi and BMW (e.g. SUV Mercedes) and we must be careful to preserve such instances in the merge result.

As discussed in Section 2 and required by property P7, two different translations for inverse-isa correspondences are possible. In this paper we present only the approach that simply combines input concepts and keeps the result more compact.

For each inv-isa-edge defined as $e = A \rightarrow B$, called, respectively, *label*(A) and *label*(B) the labels of nodes A and B, we combine A and B in a concept with label *label*(B)+

subset(A) and generate a new concept with label *label*(*A*)+ "(*others*)". The "+" operator indicates here a simple concatenation function; the *subset*() function indicates a subset of the original set when applied some special conditions. More details are in Section 4.

Let us consider the two inverse-isa-correspondences $inv-isa_1$ and $inv-isa_2$ in our example:

$$inv-isa_1 = SUV \rightarrow SUV\ Audi$$
$$inv-isa_2 = SUV \rightarrow SUV\ BMW \quad (1)$$

This step will produce the concepts *SUV Audi*+$subset_{Audi}$ (*SUV*), *SUV BMW*+$subset_{BMW}$ (*SUV*) and *SUV* (*others*), where *SUV* (*others*) represents the following set:

$$SUV(others) \equiv S \backslash (subset_{Audi}(S) \cup subset_{BMW}(S)) \quad (2)$$

where *S* is the original source concept *SUV*.

By analyzing source instances it is possible to discover if the concept *SUV(others)* is empty or not. Only if it is empty, we can remove it from the merged taxonomy. Otherwise we have to consider this node to preserve all instances.

Step 2.6 [*Finding root concept*]: Let *TLCs* be the set of all Top Level Concepts with no outgoing isa-edges. The idea is that *TLCs* contains all the candidate roots for the merge taxonomy and if *TLCs* contains only one Top Level Concept, this will be the root of $O_{T'}$, otherwise we create an artificial root node *root* in $O_{T'}$ and for each concept *c* in *TLCs* we create an is-a relationship from *c* to *root*. Note that a source TLC *c* can have, for example, an outgoing isa-edge. In this case $c \notin TLCs$ since an is-a relationship between *c* and a target concept has already been created and *c* cannot be set as root of $O_{T'}$.

The integrated taxonomy $O_{T'}$ generated by the algorithm for our running example is shown in Fig. 5(a) and it meets the requirements P1–P7 introduced in Section 2. In particular, all the target concepts (P1) and is-a relationships (P2) are also in the merged taxonomy, all the source and the target leaf nodes were translated so that all instances are preserved (P3) and no semantic overlap was introduced in the result (P5). In particular, the tree-structure of the target is maintained and no multiple inheritance is introduced in contrast to the full merge solution. While the target structure is fully preserved, the source taxonomy is only partially included since their concepts *Sedan* and *Wagon* are covered by corresponding target concepts.

The generation of mappings as discussed in Section 4 maps all input leaf concepts to the merged taxonomy and ensures that corresponding concepts in the equivalence mapping are mapped to the same merged concept in the result (property P4). Moreover, relationships defined by isa-edges are translated in the result (P6) and the concepts covered by inverse is-a correspondences are combined (P7).

Fig. 5(b) shows the taxonomy generated by our algorithm when the input mapping contains only equivalence correspondences and steps 2.3 and 2.5 are thus not applied. This result appears inferior to the result in Fig. 5(a) since not all concepts could be well placed and there is some semantic overlap due to the differences in the original taxonomies. For example the concept *Wagon BMW* should not be in a different subtree than concept *BMW*. Furthermore, there is a likely overlap between the general *SUV* concept under *Automobile** and the more specific concepts *SUV Audi* and *SUV BMW*. A merge algorithm that uses only equivalence correspondences could not better deal with such cases since the semantic relationships between these concepts have not been expressed in the provided equivalence mapping. Note that both solutions meet the requirements defined in Section 2 but a prerequisite to improve the merge result as in Fig. 5(a) is the provision of more semantic mappings between the input taxonomies.

The described algorithm allows a fully automatic merging of input taxonomies but can still benefit for manual feedback. In fact, we already recommended user interaction in Step 2.1 to resolve cycles. We might also make Step 2.4 semi-automatic by requesting user approval before declaring a source concept as not relevant and dropping it from the merge result. This way the user could force the approach to keep some concepts for completeness even at the expense of some structural redundancy. As we will show in our evaluation, relatively few source concepts are usually candidates for dropping so that the effort for interactive approval of concept dropping is expected to be low.

3.3. Multiple inheritance

While the running example is based on single inheritance, our approach can also merge input taxonomies with multiple inheritance. This is illustrated by the example shown in Fig. 6. It represents a small subset of the match scenario proposed by the Ontology Alignment Evaluation Initiative (OAEI) [1] that merges part of the subgraph describing "*Eye Muscles*" in the Mouse Anatomy with the subgraph describing a similar concept in the NCI Thesaurus.

Both input taxonomies have a corresponding root concept *Body part*, a corresponding inner concept *Muscle* and a corresponding leaf concept *Ciliary Muscle*. The shared leaf can be reached by different paths in the source and in the target; particularly it has multiple paths in the target taxonomy due to multiple inheritance.

The integrated concept graph for the example is shown in Fig. 6(b) and the merge result in Fig. 6(c). We highlight with a white background the merged concepts and with a light and dark-gray color the concepts coming only from the source or from the target, respectively, i.e. concepts not covered by a correspondence in the input mapping. For example, the source leaf concept *Iris Muscle* does not have an equivalent concept in the target but is still relevant for the merge result since it is a leaf in the source and should be preserved according to property P3. On the other hand, source concept *Tissue* is not included in the merge result since it would introduce an additional path between *Body Part* and leaf *Ciliary Muscle* and thus lead to a semantic overlap (see property P5). In fact, the number of paths for this leaf concept in the merge result remains the same than in the target taxonomy. Moreover, as we discussed in Section 3.2, the algorithm rewards the source path $\{S_5–S_3\}$ since it preserves and extends the target structuring between the concepts *Ciliary Muscle* and *Muscle*. In fact the target is-a relationship defined by T_5 is implied by is-a relationships in S_5 and S_3.

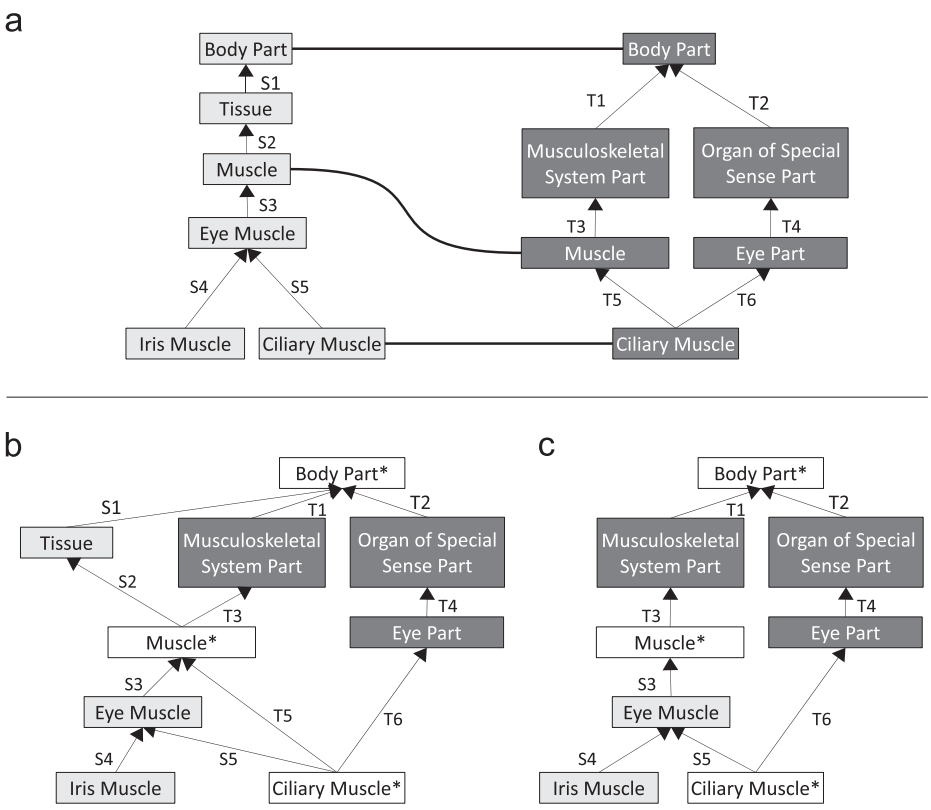

Fig. 6. Example with multiple inheritance: (a) the input taxonomies and equivalence mapping, (b) the integrated concept graph and (c) the merge result.

It is easy to see that the solution shown in Fig. 6(c) generated by the proposed algorithm meets the requirements P1–P7 introduced in Section 2 also for the example with multiple inheritance. The full merge solution is represented by the integrated concept graph shown in Fig. 6(b). It would keep the source concept *Tissue* and lead to additional paths for leaf concept *Ciliary Muscle*.

3.4. Complexity

The *complexity* of the algorithm depends on the size and on the kind of source taxonomies. We assume an average number of concepts n and an average number of is-a relationships r per input taxonomy. If O_S and O_T are taxonomies with a simple hierarchy (i.e. with no multiple inheritance), the average number of relationships is $r = n - 1$. If the input mapping contains only 1:1 correspondences, the maximum number of is-a and inverse-isa correspondences that is possible to specify is $O(n)$. The complexity of the preliminary phase is $O(r)$ and thus $O(n)$ if input taxonomies are hierarchies. If source taxonomies contain multiple inheritance, the number of edges can degenerate in the (highly unlikely) worst case to $(n(n-1))/2$ resulting in complexity $O(n^2)$. The complexity of the main phase is also different if a single or a multiple inheritance is present in the sources. In the first case the complexity is still linear with respect to the sum of source concepts, in the latter case it is quadratic with respect to source concepts.

4. Mapping generation

In this section we discuss how to automatically generate equivalence mappings between the input taxonomies and the merged taxonomy as determined by the merge algorithm. The process is fully automatic and based on the integrated concept graph reflecting the equivalence, is-a and inverse-isa relationships between the input taxonomies. These relationships produce different edges in the integrated concept graph and consequently different concepts and relationships in the merged taxonomy. In particular, equivalence correspondences describe how two concepts in the input taxonomies should be merged in the integrated taxonomy; on the other side, an isa-mapping does not produce merged concepts in the result, but it defines a subclass relationship between a source and a target concept, describing which should be the parent of a source concept in the merged taxonomy; finally, an inverse-isa-mapping describes how to split a source concept – and its instances – into two or more concepts in the final result. Algorithm 3 shows how mappings M_S and M_T for relating the input taxonomies O_S and O_T to the integrated taxonomy, respectively, are determined. The algorithm will determine in M_T a correspondence for every target concept (since the merge algorithm is target-maintaining) and in M_S a correspondence for every source concept explicitly reflected in the merged taxonomy. In particular there will be a correspondence for every leaf concept specifying where instances should migrate in the merged taxonomy.

Input equivalence correspondences and is-a correspondences can be translated at the same time analyzing nodes

marked as relevant in the integrated concept graph I. Each integrated concept in I contains the collections of source and target concepts from which it was generated. In this way, given an integrated concept C, it is always possible to know if C was present only in the source, only in the target or in both. In the following, we call, respectively, sc_c and tc_c the collections of source and target concepts for an integrated concept C. It is important to note that we assume that all integrated concepts with a nonempty tc_c are marked as relevant by default – i.e. all concepts in the target taxonomy are relevant and will be translated in the merged taxonomy.

As described in Algorithm 3, for each relevant integrated concept C_I in I such that C has no outgoing inv-isa-edges (they will be translated later in a different way), called sc_c and tc_c the sets of source and target concepts, for each concept C in sc_c, we create a correspondence between C and C_I in M_S; similarly we create a correspondence in M_T for concepts in tc_c.

For example, if we consider the integrated concept *Wagon Audi** in Fig. 3, the sets sc_c and tc_c are, respectively, $sc_{Wagon\ Audi^*} = \{Wagon\ Audi\}$ and $tc_{Wagon\ Audi^*} = \{Wagon\ Audi\}$ and we will create a correspondence between the source concept *Wagon Audi* and the integrated concept *Wagon Audi** in M_S and another one between the target concept *Wagon Audi* and *Wagon Audi** in M_T. If we consider, instead, the node *Wagon BMW*, $sc_{Wagon\ BMW} = \{Wagon\ BMW\}$ while $tc_{Wagon\ BMW}$ is empty and only one correspondence between the source concept *Wagon BMW* and the integrated concept *Wagon BMW* will be generated in M_S.

The translation of inverse-isa-correspondences is a delicate step in the mapping generation process since each correspondence describes how to move only "some" instances of a source concept and not all instances related to it. In our running example correspondences with label $inv-isa_1$ and $inv-isa_2$ state that instances in the source concept *SUV* should be split in to three disjoint subsets: *SUV Audi*, *SUV BMW* and *SUV* (*others*). As proposed in [14], a correspondence can have an attached filter – called *filter condition* – that states under which conditions the correspondence must be applied. For example, supposing the *SUV* concept has an attribute called *manufacturer*, we can define the following filter conditions, respectively, for correspondences $inv-isa_1$ and $inv-isa_2$:

$$fc_1 = [SUV.manufacturer = \text{'}Audi\text{'}]$$

$$fc_2 = [SUV.manufacturer = \text{'}BMW\text{'}] \quad (3)$$

Filter conditions can be automatically generated by a matching tool when correspondences are created or they can be manually defined by a user; as discussed for correspondences, we do not investigate how filter conditions are generated since they are part of our algorithm input. At this point, we are ready to present how inverse-isa correspondences are translated in the mapping generation process. As shown in Algorithm 3, for each node C_I with at least one outgoing inv-isa-edge, called C the corresponding source concept and called *c-edges* the list of all outgoing inv-isa-edges for C_I, we create a correspondence for each edge e in *c-edges* and we attach the filter condition fc defined on e. Finally, we create a correspondence between C and the integrated concept with label "*(others)*" attaching a filter condition fc_{others} defined as the negation of all filter conditions attached on inv-isa-edges outgoing from C_I.

If we consider again the concept *SUV* in our running example, the set of outgoing inv-isa-edges is $\{inv-isa_1, inv-isa_2\}$ with filter conditions fc_1 and fc_2 defined above. With respect to $inv-isa_1$ edge, we create a correspondence between the source concept *SUV* and the integrated concept *SUV Audi* attaching fc_1 on it; similarly for $inv-isa_2$. Finally, we create a correspondence between *SUV* and *SUV* (*others*) and we define a new filter condition as given in the following:

$$fc_{others} = [SUV.manuf\ !=\ \text{'}Audi\text{'}\ AND\ SUV.manuf\ !=\ \text{'}BMW\text{'}] \quad (4)$$

The last correspondence states that all SUV vehicles with a manufacturer different from *Audi* and *BMW* must be moved to the concept *SUV* (*others*) in the merged taxonomy.

It is important to note that if concepts in the input taxonomies have attributes, we create an attribute correspondence between each of them and the corresponding attribute in the integrated concept. We omit details of this process here.

Fig. 7 shows the mappings M_S and M_T generated for our running example presented in Fig. 1; we have drawn correspondences between leaf nodes with a solid line and that ones between inner nodes with a dotted line. It is easy to see that properties P3–P5 introduced in Section 2 are satisfied with respect to instances, since there is a correspondence for each leaf node in source taxonomies

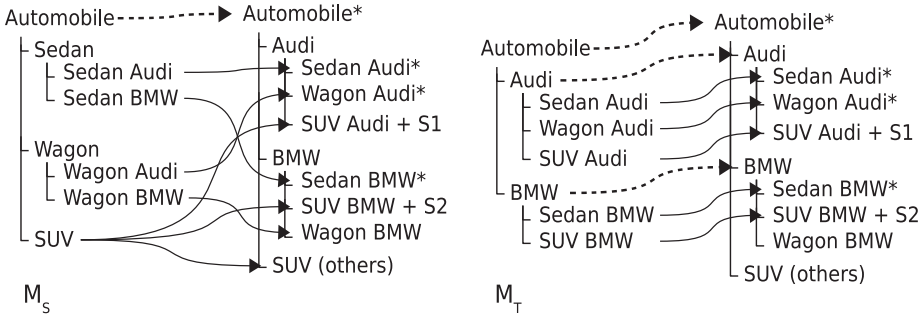

Fig. 7. Mapping M_T and M_S.

and each instance migrates to exactly one concept in the merged taxonomy; moreover if there exists an equivalence correspondence between two concepts in the input mapping, they are mapped to the same merged concept.

5. Evaluation

In this section we comparatively evaluate our approach and the full merge solution for medium- and large-sized real ontologies of two domains. The evaluation focusses on qualitative aspects related to the degree of semantic overlap (property P5) as well as the runtime efficiency.

The proposed merge algorithm has been implemented in the Atom system [26], a working prototype written in Java offering a GUI to explore all steps of generating the merged taxonomy. Atom allows the user to load input taxonomies in different formats such as OWL, XSD or XML. The mapping can be semi-automatically determined by an internal match engine or it can be provided by the user in plain text or XML format. For our evaluation we run the merge algorithms in fully automatic mode, i.e. we do not consider manual interactions to influence the merge result.

We have used the prototype to run a number of experiments both on synthetic and real-life scenarios. The synthetic scenarios were of rather small size and mainly served to verify the correctness of the merge algorithms. We therefore focus the evaluation on two real-life scenarios with ontologies of up to about 23,500 concepts. The first scenario is from the life sciences domain and aims at merging the AdultMouseAnatomy (over 2700 concepts) with the anatomical part of the NCI Thesaurus (NCIT) (about 3300 concepts). This scenario builds on the anatomy match scenario from the OAEI [1] and has the advantage that a near-perfect equivalence match mapping is available. The anatomy ontologies contain both is-a and part-of relationships as well as multiple inheritance. For our experiments, we considered only is-a relationships. For the second scenario, we merge different versions of the eBay product catalog. These versions of the catalog contain on average more than 22,000 concepts organized in a tree structure and input mappings contain more than 20,000 equivalence correspondences. Since we obtained a similar trend in all eBay experiments, we present results only for the largest scenario.

5.1. Quality of the merge solution

Evaluating the quality of an ontology merging algorithm is a challenging task since it requires evaluating the quality of the generated, integrated ontology. This quality is however strongly dependent on the quality of the input ontologies which is in turn influenced by the application purposes of the ontologies and other factors that are not under our control for real-world ontologies. Precisely determining the quality of a merge result would also require to compare it with a perfect or near-perfect merge result which is almost impossible to obtain for large ontologies or non-existent since there might be many reasonable merge results [6]. These reasons also explain why there is no benchmark that could be used to evaluate the quality of the proposed approach, e.g. by using standard quality measures, such as Precision, Recall or F-Measure.

We ensure the quality of our approach by guaranteeing that it satisfies the properties stated in Section 2. For a quantitative evaluation of quality aspects we compare the results for the two scenarios for both the full merge solution as well as the Atom approach. In particular, we consider the size or compactness of the result and evaluate the introduced degree of semantic overlap or redundancy in the merge result. For the latter aspect we especially consider the number of leaf paths in both solutions and the difference compared to the number of leaf paths in the target taxonomy (Δ leaf paths).

Table 1 summarizes our experiments; we report the rounded number of concepts and leaf paths for each input taxonomy and the number of correspondences given as input. Moreover we report the size of the solution produced by Atom and that one of the full merge solution expressed as their number of concepts, leaf concepts, total and Δ leaf paths. We remark that the full and the Atom solutions have the same number of leaves since both preserve all input leaf concepts. On the other hand, the number of leaf paths can be different in the two solutions since in Atom result overlapping concepts and relationships coming from the source taxonomy are not translated.

We observe for both experiments that Atom solutions are more compact than the full merge solutions both in terms of the number of concepts and especially in terms of the number of leaf paths. The merged taxonomy for the asymmetric Atom solution has only about 100 (inner) concepts less than for the full merge approach indicating that only very few concepts ($< 0.5\%$ of the source taxonomy) have been dropped. By contrast, Atom reduced the total number of leaf paths in the merge result by half compared to a full merge for the Anatomy scenario. We remark that the ontologies in the Anatomy scenario contain multiple inheritance so that the number of leaf paths

Table 1
Summary of experiments.

Merge example	Anatomy		eBay catalog	
	Mouse	NCI	v94	v93
Input size				
Concepts	2700	3300	21,000	23,500
Leaf paths	2300	2600	18,400	19,700
Correspondences	1500		20,200	
FULL solution				
Concepts	4500		23,400	
Leaf concepts	3500		20,400	
Leaf paths	14,000		21,500	
Δ Leaf paths	11,400		1800	
Atom solution				
Concepts	4400		23,300	
Leaf concepts	3500		20,400	
Leaf paths	7200		20,400	
Δ Leaf paths	4600		700	
Execution time (s)	1		7	

in the full solution tends to substantially increase. In fact, the number of leaf paths increases by more than a factor 4 for the full merge solution compared to the target ontology indicating that a huge degree of redundancy has been introduced. By contrast, Atom introduces 60% fewer additional leaf paths (Δ leaf paths) compared to the full merge.

We see a different result with eBay scenario reporting a reduction of the total number of leaf paths of 5%, since eBay taxonomies have a larger overlap than Anatomy ones (93% of equivalent concepts in eBay and only 50% in Anatomy scenario); on the other side, we obtain a similar relative reduction (60%) comparing Δ leaf paths, since only few inner source concepts are copied in the result producing a limited number of leaf paths that are more relevant if compared to the target.

Finally, we note that the total number of leaf paths in the Atom solution is the same as the number of leaves in the result for eBay scenario, showing that, unlike the full solution, the merged taxonomy still maintains the tree-structure of the target and no multiple inheritance is introduced.

5.2. Performance and scalability

We now analyze the runtime performance and scalability of the merge algorithm for the real-life scenarios. As already discussed in previous sections, the complexity of the algorithm is theoretically linear w.r.t. the number of concepts for taxonomies with single inheritance and quadratic in the presence of multiple inheritance. All experiments have been executed on an Intel Xeon machine with 2.66 Ghz processors, 4 GB of RAM and a 64 bit operating system. We experimentally confirmed that Atom has good performance also with real and large taxonomies. In particular, ontologies in the Anatomy scenario have multiple inheritance, while each concept in eBay product catalogs has only one parent. We measured execution time for each step in the algorithm, in particular the time necessary to generate the integrated concept graph and to produce the final result. We also measured the execution time to generate the final mappings between input and merge result. Fig. 8 shows the detailed execution times for the two medium and large-sized scenarios indicating that generating the integrated concept graphs is the most expensive step. The system generated the final result in less than 1 s for the medium-size scenario with multiple inheritance (Mouse-NCI), and in about 7 s for the large-sized one (eBay product catalog), showing a very good scalability of the algorithm.

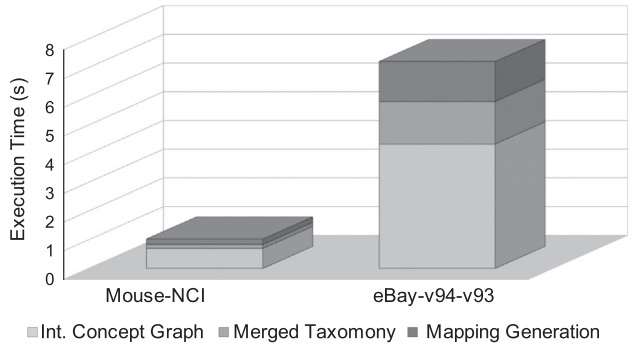

Fig. 8. Execution times on large-scale scenarios.

6. Related work

Integrating heterogeneous ontologies or other metadata models such as database schemas are inherently complex problems that have been investigated in research since several decades [2,16]. Early schema integration approaches tried to come up with comprehensive solutions covering all related steps such as determining corresponding schema elements, resolving conflicting representations as well as merging and combining the information from different input schemas. The problem with such approaches is that they tend to become very complex and difficult to use, even when they focus on specific schema languages or application domains [3]. The more recent approaches to schema and ontology integration have thus mostly followed a more modular scheme by decomposing the integration problem into independently solvable subproblems such as Match and Merge. The Match step is responsible for aligning two or more ontologies or schemas. It produces a mapping specifying all corresponding concepts or schema elements. This mapping can then be used by the Merge step to determine the integrated ontology or integrated schema.

The vast majority of research in the last 10–15 years on schema and ontology integration has focussed on the match step, i.e. schema matching and ontology alignment. As described in several surveys and books (e.g., [25,8,24]), numerous approaches and prototypes have been developed for ontology matching that typically utilize the linguistic and structural similarity of ontology concepts; some approaches also consider the similarity of ontology instances. Most approaches focus on determining an equivalence mapping containing all pairs of semantically equivalent ontology concepts. Several approaches have also been developed to determine more general ontology mappings, including is-a relationships between ontologies [30,31,34]. Match approaches are typically semi-automatic, i.e. their result needs to be verified and corrected (deletion of wrong correspondences, addition of missed correspondences) by a human domain expert to obtain the correct mapping.

The previous approaches to merge two schemas or two ontologies can be classified as shown in Fig. 9. We use three largely orthogonal criteria namely whether the merge approaches are symmetric or asymmetric, whether or not they are based on an input mapping and whether the approach focusses on (database) schema or ontology merging. Some approaches, e.g. “Vanilla” [22], are meant to be generic, i.e. apply to both schemas and ontologies.

Merging of schemas has first been addressed within comprehensive, largely manual approaches for schema integration [2]. MOMIS is a well-known representative of such systems [4]. More recent schema merging approaches are “mapping-based”, i.e. they assume a correct input mapping specifying all correspondences between the input schemas. Such a mapping is typically the result of a semi-automatic Match step and specifies the

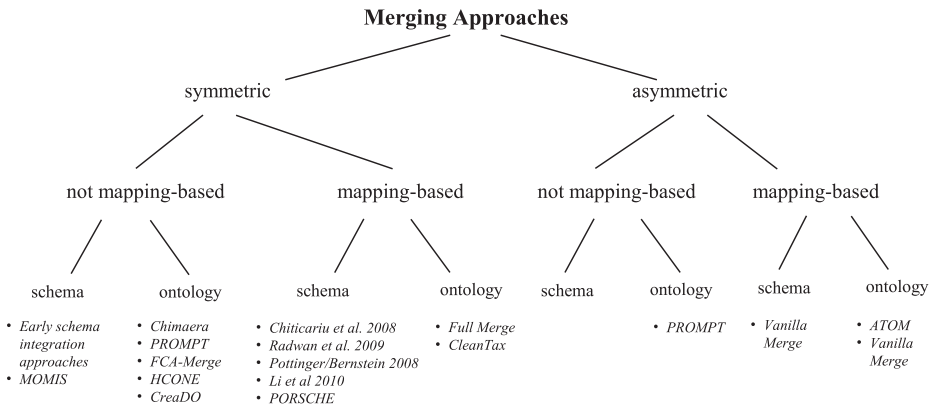

Fig. 9. Classification of merging approaches.

corresponding or related information to be integrated. The Merge step can thus significantly be reduced in complexity and much better automated [20]. Pottinger provides a recent overview about mapping-based merge approaches [19]. One of the first studies on mapping-based merge approaches [22], specified general requirements for schema merging and proposed a generic "Vanilla" implementation. As discussed in Section 2 we reuse and extend some of their requirements adapting them to our target-driven algorithm.

Most previous mapping-based schema merge approaches are symmetric and try to completely preserve the information of the input schemas [6,28,20,29,23,9,21]. The approach of Chiticariu et al. [6] generates numerous integrated candidate schemas and requires substantial user intervention to iteratively refine the result. We discussed the differences between [6] and our approach in Section 3. Radwan et al. [23] build on [6] but propose a more automatic approach that uses directed and weighted correspondences to rank the integrated schemas for simplified user selection. The approaches proposed in by Pottinger and Bernstein [20] as well as by Li et al. [12] take as input a set of relational schema and an equivalence mapping expressed under the form of conjunctive queries. The PORSCHE system [28] focusses on the merging of XML schemas based on previously determined equivalence correspondences.

For ontology merging, virtually all previous approaches have been symmetric. A notable exception is PROMPT that can be used symmetrically (see below) or asymmetrically. Support for asymmetric merge is provided by PROMPT since one can specify a preferred ontology that helps to reduce the number of user interactions. ATOM is the first asymmetric and mapping-based ontology merging scheme that tries to reduce the semantic overlap in the merge result through its target-driven approach. Furthermore, it is the first mapping-based merge approach to use more semantic match mappings including is-a or inverse-is-a correspondences, to improve the merge result.

For ontology merging, well-known approaches such as PROMPT, Chimaera and FCA-Merge [17,13,32] are symmetric and not mapping-based. These approaches primarily focus on the problem of ontology alignment and do not use the separation of matching and merging. The merging algorithms of both PROMPT [17] and Chimaera [13] are semi-automatic performing some tasks automatically but relying on user input for other tasks. PROMPT checks whether such user interventions generate inconsistencies in the merged ontology and suggests possible solutions to reach a consistent ontology. The merge algorithm of FCA-Merge [32] is based on a lattice of concepts. The lattice is automatically derived but the generation of the merge ontology requires the user intervention to explore the lattice.

Further semi-automatic, symmetric ontology merging approaches include HCONE [10] and CreaDo [18]. Both depend on dictionaries such as WordNet for matching and are thus currently limited to knowledge represented in English language. CreaDo aims at limiting the scope of the merged ontology for improved understandability by restricting it to a significant concept (to be provided as a parameter) and the related portions from the source ontologies that can be mapped to the specified concept [18].

CleanTax is one of the few mapping-based ontology merging approaches that, like we, limits itself to taxonomies [33]. They investigate the merging problem when the relationships between concepts of different taxonomies can be expressed as algebraic (RCC-5) constraints. The simple "Full merge" approach considered in this paper is also a scheme for mapping-based taxonomy merging.

Relatively little work has been performed on the evaluation of schema and ontology merging. Lambrix qualitatively compared the PROMPT and Chimaera tools for merging biomedical ontologies [11]. However, the focus was not on the merge result but on the system itself and the user effort to obtain a specific result. In [7], an approach to evaluate the quality of an integrated schema (i.e. the result of schema merging approach) is proposed. They require the specification of a perfect integrated schema and define metrics such as *Structurality*, *Completeness* and *Minimality* to assess the proximity of an integrated schema to the (presumably) perfect one. As already discussed in Section 5, more than one ideal solution could exist and, moreover, it is hardly possible to manually generate an ideal merged schema for large scenarios. In a recent paper [27], we have advocated for the use of simple criteria to comparatively evaluate the quality of different ontology merging approaches similarly as we have done it in our evaluation here.

When citing this as related work: Euler uses RCC5, so we have in addition OVERLAPS (very important in practice) and DISJOINT (also useful for many inferences).

7. Conclusions

We proposed the ATOM approach for automatically merging a source taxonomy into a target taxonomy. The merge algorithm is target-driven and preserves all concepts and relationships of the target taxonomy and can largely limit the semantic overlap in the merged taxonomy. Furthermore, ATOM is based on a match input mapping and can utilize not only equivalence correspondences but also is-a and inverse-isa correspondences to improve the merge result. ATOM also determines mappings between the input taxonomies and the merge result that can be used for instance migration. The proposed algorithms have linear complexity for hierarchical taxonomies. The ATOM approach could be successfully applied to large real-life taxonomies from different domains. ATOM generates a default solution in a fully automatic way that may interactively be adapted by users if needed.

There are several opportunities for future work. First it is desirable to investigate how the asymmetric merge approach can be extended for more general ontologies without introducing the need for substantially more manual interaction. Second, more work to evaluate the quality of merged ontologies and merge algorithms is needed, e.g. by focussing on different kinds of applications and their interoperability requirements. Furthermore, it would be interesting to investigate the potential of asymmetric merge for integrating more than two ontologies.

Appendix A. Algorithms and pseudo-code.

Algorithm 1. *ICGGen*($G_S, G_T, Corr$).

Input: two concept graphs $G_S=(V_S,E_S)$ and $G_T=(V_T,E_T)$ and a set of correspondences *Corr*
Output: an Integrated Concept Graph *I*

$I=(V,E)\leftarrow$ empty
for each node x in $V_S \cup V_T$ **do**
  if x is involved in an equivalence correspondence (x,y) or (y,x) **then**
    $c\leftarrow$ merge concepts x and y
    add c to V
  else
    add x to V
  end if
end for
for each edge e_S in E_S **do**
  generate a *s-edge* in *I*
end for
for each edge e_T in E_T **do**
  generate a *t-edge* in *I*
end for
for each is-a correspondence c_{isa} in *Corr* **do**
  generate an *isa-edge* in *I*
end for
for each inverse-isa correspondence c_{inv} in *Corr* **do**
  generate an *inv-isa-edge* in *I*
end for
return *I*

Algorithm 2. *IntegratedTaxGen*(I).

Input: an Integrated Concept Graph $I=(V,E)$
Output: an Integrated Taxonomy $O_{T'}$

comment Step 2.1
RemoveCycles(*I*)
comment: Step 2.2
for each t-edge $e=N_1\rightarrow N_2$ in *I* **do**
  if $\exists$ exactly one c-path *P* from N_1 to N_2 s.t. $length(P)>1$ **then**
    mark all edges in *P* as *relevant*
  else
    create an is-a relationship between N_1 and N_2 in $O_{T'}$
  end if
end for
comment: Step 2.3
for each isa-edge $e=N_1\rightarrow N_2$ in *I* **do**
  create an is-a relationship between N_1 and N_2 in $O_{T'}$
end for
comment: Step 2.4
$candidates\leftarrow\{X : X\in V \wedge X$ has at least one outgoing s-edge but no incoming s-edges$\}$
for each *X* in *candidates* **do**
  spaths$\leftarrow$ set of s-paths with start node *X*
  for each s-path *P* in *spaths* **do**
    for each edge $e=C_1\rightarrow C_2$ in *P* **do**
      if C_1 has no outgoing t-edges or isa-edges **then**
        mark *e* as *relevant*
      else
        break
      end if
    end for
  end for
end for
for each relevant edge $e=N_1\rightarrow N_2$ **do**
  create an is-a relationship between N_1 and N_2 in $O_{T'}$
end for
comment: Step 2.5
for each inv-isa-edge $e=N_1\rightarrow N_2$ in *I* **do**
  if exists a relevant s-edge or t-edge $r=N_2\rightarrow N_1$ **then**
    create a new concept N_{others} with $label(N_{others})\leftarrow label(N_1)+$"(*others*)" (if does not exist a similar concept)
    create a new is-a relationship between N_{others} and N_1
  else
    if N_1 has not yet been renamed **then**
      $label(N_1)\leftarrow label(N_1)+$"(*others*)"
    end if
    $label(N_2)\leftarrow label(N_2)+subset(N_1)$
  end if
end for
comment: Step 2.6
$A\leftarrow$ set of all *relevant* source TLC in *I* with no outgoing is-a edges
$B\leftarrow$ set of all target top level concepts in *I*
$TLCs\leftarrow A\cup B$
if $size(TLCs)=1$ **then**
  $root(T)\leftarrow TLCs[0]$
else
  $root(T)\leftarrow$ create a new node *R*
  for each top level concept *tlc* in *TLCs* **do**
    nest *tlc* in $root(T)$
  end for
end if
return $O_{T'}$

Algorithm 3. *MappingGen*(I, T).

Input: an Integrated Concept Graph *I* and a merged taxonomy $O_{T'}$
Output: two equivalence mappings M_S and M_T

$M_s\leftarrow$ empty
$M_t\leftarrow$ empty
$E_0\leftarrow$ all merged nodes in *I*
$E_1\leftarrow$ relevant nodes in *I* with no outgoing inv-isa-edges
$E_2\leftarrow$ nodes in *I* with at least one outgoing inv-isa-edge
for each node C_I in $E_0\cup E_1$ **do**
  $sc\leftarrow$ set of source concepts for C_I
  $tc\leftarrow$ set of target concepts for C_I
  for each concept *C* in *tc* **do**
    create a correspondence $C-C_I$ in M_T
  end for
  for each concept *C* in *sc* **do**

```
13:       create a correspondence C − C_I in M_S
14:     end for
15:   end for
16:   for each node C_I in E_2 do
17:     fcs ← empty
18:     c_edges ← set of all inv-isa-edges outgoing from C_I
19:     for each edge e = C_1 → C_2 in c_edges do
20:       create a corr. C_2 − C_I in M_s with a filter cond. fc_e
21:       add fc_e to fcs
22:     end for
23:     create a corr. C_1 − C_1(others) in M_S with a filter cond. "not fcs"
24:   end for
25:   return M_S, M_T
```

References

[1] The Ontology Alignment Evaluation Initiative. ⟨http://oaei.ontologymatching.org⟩.

[2] C. Batini, M. Lenzerini, S.B. Navathe, A comparative analysis of methodologies for database schema integration, ACM Comput. Surv. 18(4) (1996) 323–364.

[3] Z. Bellahsene, A. Bonifati, E. Rahm (Eds.), Schema Matching and Mapping, Springer, 2011.

[4] S. Bergamaschi, S. Castano, M. Vincini, D. Beneventano, Semantic integration of heterogeneous information sources, Data Knowl. Eng. 36 (3) (2001) 215–249.

[5] O. Bodenreider, The unified medical language system (umls): integrating biomedical terminology, Nucl. Acids Res. 32 (Suppl. 1) (2004).

[6] L. Chiticariu, P.G. Kolaitis, L. Popa, Interactive generation of integrated schemas, in: Proceedings of the ACM SIGMOD, 2008, pp. 833–846.

[7] F. Duchateau, Z. Bellahsene, Measuring the quality of an integrated schema, in: ER, 2010, pp. 261–273.

[8] J. Euzenat, P. Shvaiko, Ontology Matching, Springer-Verlag, New York, 2007.

[9] G. Ganapathy, R. Lourdusamy, Matching and merging of ontologies using conceptual graphs, in: Proceedings of the World Congress on Engineering, 2011.

[10] K. Kotis, G.A. Vouros, K. Stergiou, Towards automatic merging of domain ontologies: the hcone-merge approach, J. Web Sem. 4 (1) (2006) 60–79.

[11] P. Lambrix, A. Edberg, Evaluation of ontology merging tools in bioinformatics, in: Pacific Symposium on Biocomputing, 2003, pp. 589–600.

[12] X. Li, C. Quix, D. Kensche, S. Geisler, Automatic schema merging using mapping constraints among incomplete sources, in: International Conference on Information and Knowledge Management (CIKM), 2010, pp. 299–308.

[13] D.L. McGuinness, R. Fikes, J. Rice, S. Wilder, An environment for merging and testing large ontologies, in: KR, 2000, pp. 483–493.

[14] G. Mecca, P. Papotti, S. Raunich, Core Schema Mappings, in: Proceedings of the ACM SIGMOD, 2009, pp. 655–668.

[15] C. Mungall, et al., Uberon, an integrative multi-species anatomy ontology, Genome Biol. 13 (1) (2012).

[16] N.F. Noy, Semantic integration: a survey of ontology-based approaches, SIGMOD Rec. 33 (4) (2004) 65–70.

[17] N.F. Noy, M.A. Musen, Prompt: algorithm and tool for automated ontology merging and alignment, in: AAAI/IAAI, 2000, pp. 450–455.

[18] S. Pariente Juarez, H. Esquivel, A. Rebollar, M. Suarez-Figueroa, Creado—a methodology to create domain ontologies using parameter-based ontology merging techniques, in: Proceedings of the 10th Mexican International Conference on Artificial Intelligence (MICAI), 2011, pp. 23 –28.

[19] R. Pottinger, Mapping-based merging of schemas, in: Schema Matching and Mapping, Springer-Verlag, 2011, pp. 223–249.

[20] R. Pottinger, P.A. Bernstein, Schema merging and mapping creation for relational sources, in: International Conference on Extending Database Technology (EDBT), 2008, pp. 73–84.

[21] R. Pottinger, P.A. Bernstein, Associativity and commutativity in generic merge, in: A.T. Borgida, V.K. Chaudhri, P. Giorgini, E.S. Yu (Eds.), Conceptual Modeling, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 254–272.

[22] R.A. Pottinger, P.A. Bernstein, Merging models based on given correspondences, in: Proceedings of the 29th International Conference on Very Large Data Bases, vol. 29, VLDB Endowment, 2003, pp. 862–873.

[23] A. Radwan, L. Popa, I.R. Stanoi, A.A. Younis, Top-k generation of integrated schemas based on directed and weighted correspondences, in: SIGMOD Conference, 2009, pp. 641–654.

[24] E. Rahm, Towards Large Scale Schema and Ontology Matching, in: Schema Matching and Mapping, Springer, 2011, pp. 3–27 (Chapter 1).

[25] E. Rahm, P.A. Bernstein, A survey of approaches to automatic schema matching, in: International Journal on Very Large Data Base, vol. 10, 2001, pp. 334–350.

[26] S. Raunich, E. Rahm, ATOM: Automatic Target-driven Ontology Merging, in: ICDE, 2011, pp. 1276–1279, Demo paper.

[27] S. Raunich, E. Rahm, Towards a benchmark for ontology merging, in: OTM Workshops, 2012, pp. 124–133.

[28] K. Saleem, Z. Bellahsene, E. Hunt, Porsche: performance oriented schema mediation, Inf. Syst. 33 (7–8) (2008) 637–657.

[29] A.D. Sarma, X. Dong, A.Y. Halevy, Bootstrapping pay-as-you-go data integration systems, in: SIGMOD Conference, 2008, pp. 861–874.

[30] P. Shvaiko, F. Giunchiglia, M. Yatskevich, Semantic matching with s-match, in: Semantic Web Information Management, 2009, pp. 183–202.

[31] V. Spiliopoulos, G.A. Vouros, V. Karkaletsis, On the discovery of subsumption relations for the alignment of ontologies, J. Web Sem. 8 (1) (2010) 69–88.

[32] G. Stumme, A. Maedche, Fca-merge: Bottom-up merging of ontologies, in: IJCAI, 2001, pp. 225–234.

[33] D. Thau, S. Bowers, B. Ludäscher, Merging taxonomies under rcc-5 algebraic articulations, in: ONISW, 2008, pp. 47–54.

[34] P. Arnold, E. Rahm, Semantic enrichment of ontology mappings: a linguistic-based approach, ADBIS (2013) 42–55.