# Algebraic Graph Transformations for Merging Ontologies

Mariem Mahfoudh, Laurent Thiry, Germain Forestier, Michel Hassenforder

# Algebraic Graph Transformations for Merging Ontologies

Mariem Mahfoudh, Laurent Thiry, Germain Forestier, and Michel Hassenforder

MIPS EA 2332, Université de Haute Alsace
12 rue des Frères Lumière 68093 Mulhouse, France
{mariem.mahfoudh,laurent.thiry,germain.forestier,michel.hassenforder}@uha.fr

**Abstract.** The conception of an ontology is a complex task influenced by numerous factors like the point of view of the authors or the level of details. Consequently, several ontologies have been developed to model identical or related domains leading to partially overlapping representations. This divergence of conceptualization requires the study of ontologies merging in order to create a common repository of knowledge and integrate various sources of information. In this paper, we propose a formal approach for merging ontologies using typed graph grammars. This method relies on the algebraic approach to graph transformations, SPO (Simple PushOut) which allows a formal representation and ensures the consistence of the results. Furthermore, a new ontologies merging algorithm called GROM (Graph Rewriting for Ontology Merging) is presented.

**Keywords:** Ontologies Merging, Typed Graph Grammars, Algebraic Graph Transformations, GROM.

## 1 Introduction

With the emergence of ontologies [1] and their wider use, several ontologies have been developed to model identical or related domains leading to partially overlapping representations. As an example, we can cite the domain of Large Biomedical Ontologies which contains more than 370[1] ontologies with some famous ontologies : Foundational Model of Anatomy (FMA) [2], SNOMED CT[2], National Cancer Institute Thesaurus[3] (NCI), etc. This multitude of ontologies motivates the study of their merging to integrate and compose the different sources of knowledge.

Merging ontologies becomes more and more necessary and represents an important area of research. It is defined by Klein [3] as *"Creating a new ontology from two or more existing ontologies with overlapping parts, which can be either*

[1] http://bioportal.bioontology.org
[2] http://www.ihtsdo.org/snomed-ct
[3] http://ncit.nci.nih.gov

This is the author's version of an article published at MEDI 2014. The final authenticated version is available online at: https://doi.org/10.1007/978-3-319-11587-0_16

*virtual or physical".* The creation of the new ontology (also called the *global ontology*) is generally a complex task and requires considerable adaptation and a rigorous formalism to control the various steps of the construction. In this context, this paper proposes a formal approach for merging ontologies using typed graph grammars with algebraic graph transformations. Typed Graph Grammars ($TGG$) are a mathematical formalism which permits to represent and manage graphs. They are used in several fields of computer science such as software systems modelling, pattern recognition and formal language theory [4]. Recently, they started to be used in the ontology field, in particular for the formalization of the operations on ontologies like the alignment, merge and evolution [5,6,7,8,9,10]. In our previous work [10], we used $TGG$ to formalize and implement ontology changes. They allow, thanks to their application conditions, to control the evolution process and to avoid inconsistencies.

In this paper, we use the same formalism to describe a formal approach of ontologies merging. The proposed approach has been implemented and a new tool called GROM (Graph Rewriting for Ontology Merging) is introduced. An application is presented on two ontologies developed in the frame of the CCAlps European project. Thus, the main contribution of this work is to take advantage of the graph grammars domain and the algebraic graph transformations to define and implement the process of merging ontologies.

The rest of this paper is structured as follows: Section 2 presents an overview of the typed graph grammars and algebraic graph transformations. Section 3 proposes an approach of ontology merging. Section 4 presents an example of application. Section 5 discusses some properties of the proposed approach. Section 6 shows some related work. Finally, a conclusion summarizes the presented work and gives some perspectives.

## 2 Typed Graph Grammars

This section reviews the fundamental notions involved in typed graph grammars and algebraic graph transformations.

**Definition 1** (Typed graph grammars). A typed graph grammar is a formalism that is composed of a type graph ($TG$), a start graph ($G$ also called host graph) and a set of production rules ($P$) called graph rewriting rules (or graph transformations). In this article, we consider the typed attributed graphs. Thus, $TGG = (G, TG, P)$ where:

- $G = (N, E, src : E \rightarrow N, tgt : N \rightarrow E, att : N \cup E \rightarrow \mathcal{P}(att))$ is a graph composed of : 1) a set of nodes (N); 2) a set of edges ($E$); 3) two functions, $src$ and $tgt$, which specify the source and target of an edge; 4) a set of attributes ($att$) which are associated to the edges and nodes.
- $TG = (N_T, E_T, src : E_T \rightarrow N_T, tgt : N_T \rightarrow E_T, att_T)$ is a graph which represents the type of the elements of the graph $G$. The typing of a graph G over $TG$ is given by a total graph morphism $t : G \rightarrow TG$ defined by 3 functions $t_E : E \rightarrow E_T$, $t_N : N \rightarrow N_T$ and $t_{att} : att \rightarrow att_T$. Figure 1 shows

an example of type graph and host graph. The $TG$ represents two nodes "Conference" and "Emplacement" which have respectively two attributes "name" and "description" and linked by an edge "hasPlace". The host graph $G$ represents an instance of the $TG$.

– $P$ is a set of production rules which permit the replacement of one sub-graph by another. It is defined by a pair of graphs patterns ($LHS$, $RHS$) where:
  - $LHS$ (Left Hand Side) represents the preconditions of the rewriting rule and describes the structure that has to be found in $G$.
  - $RHS$ (Right Hand Side) represents the postconditions of the rule and must replaces $LHS$ in $G$.

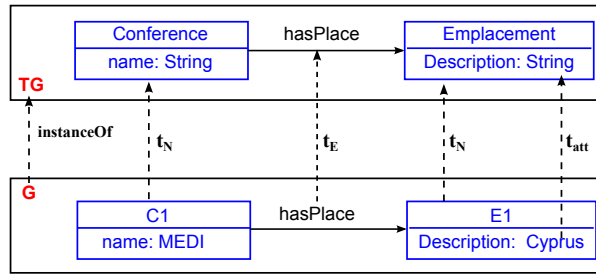



Fig. 1: Example of Type Graph and Host Graph.

Moreover, the rules are allowed to have negative application conditions ($NACs$). A $NAC$ is another graph pattern that specify the graph that should not occur when matching a rule. This means that rewriting rule cannot be applied if $NAC$ exists in $G$. In this way, a graph transformation defines how a graph $G$ can be transformed to a new graph $G'$. More precisely, there must exist a morphism that replaces $LHS$ by $RHS$ to obtain $G'$. To apply this replacement, different graph transformations approaches are proposed [11]. In this work, we use the algebraic approach [12] based on Category Theory [13] with the *pushout* concept.

**Definition 2** (Category Theory). A category [14] is a structure consisting of: 1) a collection of objects $O$; 2) a set of arrows (also called morphism $M$) and a function $s : M \to O \times O$ such as $s(f) = (A, B)$ is written $f : A \to B$; 3) a binary operation called composition of morphisms $(\circ) : M \times M \to M$; 4) an identity morphism for each object $id : O \to O$. The composition operator is associative and $id(O)$ is the neutral element, i.e. if $m : A \to B$ then $m \circ id(A) = m = id(B) \circ m$. In our work, we consider the category of *Graph* where the objects are the graphs and the arrows are the graph morphisms.

**Definition 3** (Pushout). Given three objects $A$, $B$ and $C$ and two morphisms $f : A \to B$ and $g : A \to C$, the pushout of $B$ and $C$ consists of: 1) an object $D$

and two morphisms $f' : B \to D$ and $g' : C \to D$ where $f' \circ f = g' \circ g$; 2) for any morphisms $f'' : B \to E$ and $g'' : C \to E$ such that $f \circ f'' = g \circ g''$, there is a unique morphism $k : D \to E$ such that $f' \circ k = f''$ and $g' \circ k = g''$.

Algebraic approaches are divided into two categories: the *Single PushOut, SPO* [15] and the *Double PushOut, DPO* [16]. Applying a rewriting rule to an initial graph ($G$) with the SPO method consists in (Figure 2):

1. Finding a matching of $LHS$ in $G$ by defining a morphism $m : LHS \to G$.
2. Deleting $m(LHS) - m(LHS \cap RHS)$ from $G$.
3. Adding $m(RHS) - m(LHS \cap RHS)$ to $G$ to give new version $G'$.

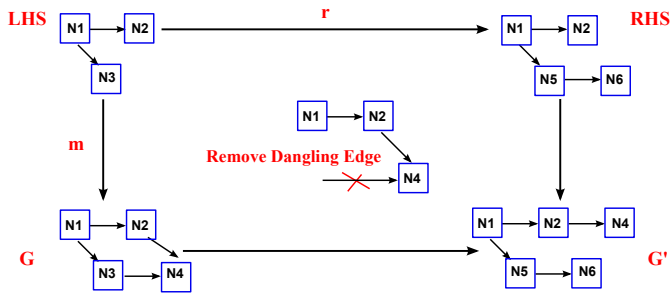



Fig. 2: Graph rewriting rule with SPO.

The DPO approach consists of two pushouts and requires an additional condition called the dangling condition. This condition states that the transformation is applicable only if its application will not lead to dangling edges. For example, for the host graph $G$ of the Figure 2, the rewriting rule is forbidden by the DPO approach because it breaks the dangling condition. If we want to apply this rule, the host graph $G$ should not contain the edge $E(N3, N4)$. In the SPO approach, the dangling edges are removed. This allows to write transformations that do not allow DPO approach which is limited by the dangling condition. For this reason, we only consider the SPO approach in this work.

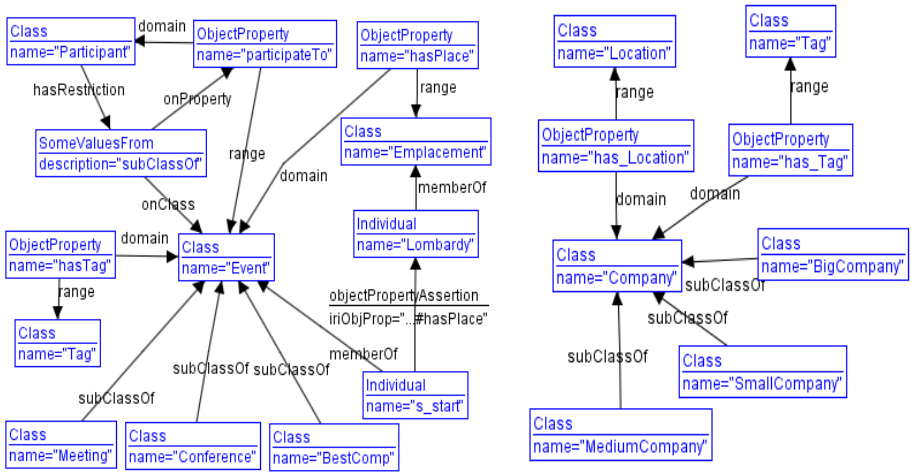
## 3 Merging ontologies with Typed Graph Grammar

### 3.1 Ontologies with Typed Graph Grammars

As mentioned above, a typed graph grammar is defined by $TGG = (TG, G, P)$. By adapting this definition to the ontology field, we obtain:

– $G$ is the host graph which represents the ontology. Figure 3 shows two examples of host graphs. They are sub-ontologies from the EOCCAlps (Event Ontology CCAlps) and COCCAlps (Company Ontology CCAlps) which are developed in the frame of the European project CCAlps[4]. The EOCCAlps

[4] http://www.ccalps.eu




(a) Extract from EOCCAlps ontology ($O_1$)

(b) Extract from COCCAlps ontology ($O_2$)

Fig. 3: Example of ontologies represented as a host graph.

ontology represents events. They can be a "Conference", "Meeting" or a "BestComp" and should takes place in the Alpine space. The COCCAlps ontology represents companies and is used to describe those which will be participate to the events.

– $TG$ is the type graph which represents the meta-model of the ontology (see [10]). The OWL meta-model was chosen because it is the standard proposed by the W3C and the language usually adopted to represent ontologies. Thus, the types of nodes, $N_T = \{Class, Property, Individual, DataType, Restriction\}$ and the type of edges are the axioms, $E_T = \{subClassOf, domain, ...\}$.
  - Classes ($C$) model the set of individuals. For example, for EOCCAlps ontology, $C = \{\text{“}Event\text{”}, \text{“}Meeting\text{”}, \text{“}Participant\text{”}, ...\}$.
  - Individuals ($I$) represent the instances of classes, $I = \{\text{“}Lombardy\text{”}, \text{“}It's-Start\text{”}\}$.
  - Properties ($P$), for each class $C_i \in C$, there exists a set of properties $P(C_i) = DP(C_i) \cup OP(C_i)$, where $DP$ are datatype properties and $OP$ are object properties. If a property relates a class $C_i$ to an entity $E_i$ ($Class$ or $Datatype$), then $C_i$ is called the *domain* of the property and $E_i$ is called the *range* of the property. For example, $OP = \{\text{“}hasTag\text{”}\}$, $domain(\text{“}hasTag\text{”}) = \{\text{“}Event\text{”}\}$, $range(\text{“}hasTag\text{”}) = \{\text{“}Tag\text{”}\}$.
  - Datatypes ($D$) represent the type of data value. They can be string, boolean, etc.
  - Restrictions ($R$), for each property $p \in P(C_i)$ there exists a set of restrictions on the value or cardinality. For example, there is a value restriction on the property "participateTo" which states that some value for the "participateTo" should be an instance of the class "Participant".

- Axioms ($A$) specify the relations between the ontologies entities. For example, $subClassOf$ represents the subsumption relation between classes.

Note that both nodes and the edges can contain attributes. For example, among the attributes of the nodes of types $C$, $I$ and $P$, we found the attribute *name* which specifies their local names and the *iri* which identifies them. In the figures of this article, the *iri* has not represented for readability reasons.

– $P$ are the rewriting rules corresponding to the ontology changes ($AddClass$, $RemoveDataProperty$, $RenameIndividual$, etc.). An ontology change is defined by $CH = (Name, NACs, LHS, RHS, DCHs)$ where: 1) $Name$ specifies the type of change; 2) $NACs$ define the conditions which must not be true to apply the rewriting rule; 3) $LHS$ represents the precondition of the rewriting rule; 4) $RHS$ defines the postcondition of the rewriting rule; 5) $DCHs$ are the Derived CHanges. They are additional rewriting rules that are attached to $CH$ to correct its possible inconsistencies.
Figure 4 shows an example of a rewriting rule for the $RenameClass$ change. This rule renames a node of type $Class$ "Company" to "Enterprise" while avoiding redundant elements by the $NAC$.

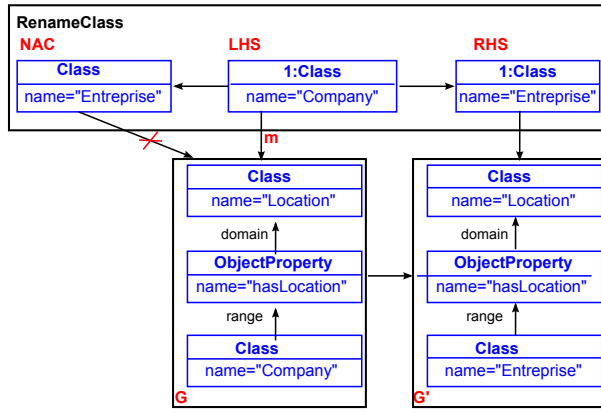



Fig. 4: Rewriting rule for the $RenameClass$ change.

After introducing how to use typed graph grammars to represent ontologies and their changes, we present now an approach for merging ontologies. This consists mainly of three steps presented in the following sections: 3.2) similarity search; 3.3) merging ontologies; 3.4) global ontology adaptation (Figure 5).

### 3.2 Similarity Search

In order to establish a correspondence between the ontologies, it is necessary to identify the relationship (similarity) between their entities. In the literature,

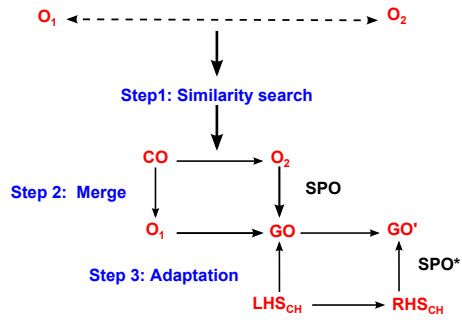



Fig. 5: Approach of merging ontologies with algebraic graph transformations.

several techniques have been proposed to determine these similarities [17]. They can be divided into five categories:

1. lexical techniques, consider the name of the entities and compare them as String, e.g. Levenshtein distance [18];
2. structural techniques, consider the structure of the ontologies to detect the subsumption relations, e.g. Children and Leaves [19];
3. strategies that use an external resource like ontology linguistic (e.g. Wordnet [20]), dictionary, thesaurus or other ontology for the same domain;
4. strategies that compare ontologies through their usage traces (ex. annotations of the same resources);
5. combination of the previous techniques.

Despite this multitude of techniques, the domain of similarity search has still several challenges [21]. Thus, considering that our main goal is merging ontologies, we chose to work with a simple, but efficient, combination of lexical techniques and external resource. Thus, Levenshtein distance is used for detecting the common and equivalent terms and WordNet is used to recognize the semantic correspondences essentially the synonym terms. The subsumption relations ($IsaN$) are defined manually. Then, the process of the similarity search takes two ontologies ($O_1$ and $O_2$) as input and compares their components (class by class, property by property and individual by individual). Then, it generates:

- $CN = \{N_i | (N_i \in N(O_1)) \wedge (\exists N_j \in N(O_2) \cdot (N_{iT} = N_{jT}) \wedge (Levenshtein(N_i.name, N_j.name) = 0))\}$ is the set of common nodes between the nodes of ontology $O_1$ ($N(O_1)$) and those of ontology $O_2$ ($N(O_2)$) where the type of nodes can be $Class$, $Property$ ($DP$ or $OP$) or $Individual$;
- $EN = \{(N_i, N_j) | (N_i \in N(O_1)) \wedge (N_j \in N(O_2)) \wedge (N_{iT} = N_{jT}) \wedge (Levenshtein(N_i.name, N_j.name) < threshold))\}$ is the set of the equivalent nodes;
- $SN = \{(N_i, N_j) | (N_i \in N(O_1)) \wedge (N_j \in N(O_2)) \wedge (N_{iT} = N_{jT}) \wedge (N_i.name \in (synsetWordNet(N_j.name)\}$ is the set of the nodes sharing a semantic relation.

### 3.3 Merging Ontologies

The process of ontologies merging is based on SPO approach which offers a rigorous and simple way to glue the graphs. It encapsulates the complex details of the ontologies structures by considering them as objects of a suitable abstract category. Thus, this step is divided into three sub-steps (see Figure 6). The first one aims at minimizing the differences between the two ontologies. Thus, its role is to replace the entities of the ontology 1 by their equivalent in the ontology 2. This replacement is applied by the rewriting rule $RenameNode$ ($N_i$, $N_j$) where $N_i$ is a node of $O_1$ and $N_j$ is its equivalent in $O_2$. So, for this SPO:

- the host graph is the ontology $O_1$;
- the $LHS$ is the graph composed by the set of nodes $\{N_i \in EN\}$;
- the $RHS$ is the graph composed by $\{N_j \in EN\}$.

An example of the $RenameClass$ rule is already presented in Figure 4.

Then, the second step consists in creating the Common Ontology ($CO$). This is the common sub-graph between the two ontologies. It is constructed by the common nodes ($CN$) and the edges that they share.

The third and last step merges the ontologies with the $MergeGraph(CO, O_2)$ rewriting rule. This SPO is defined as following:

- its host graph is the ontology $O_1$ after modification ($O'_1$);
- its $LHS$ is the Common Ontology $CO$;
- its $RHS$ is the ontology $O_2$.

The role of this pushout is the merge of the two ontologies by linking them by their common entities. Thus, this step provides a global ontology ($GO$) which will be enriched by the integration of semantic and subsumption relations.

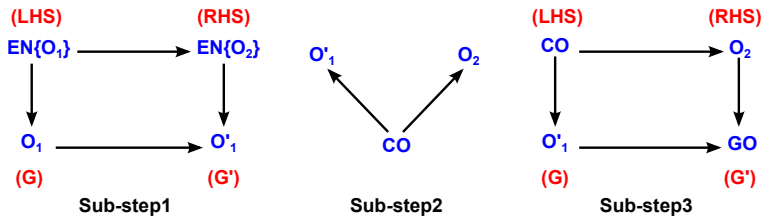



Fig. 6: The three sub-steps of the ontologies merging.

### 3.4 Global Ontology Adaptation

Given that the graph transformation requires the presence of the match (i.e a morphism $m$) between $LHS$ and the host graph, the addition of the semantic and

the subsumption relations should be applied after the creation of the global ontology. Thus, this section presents the *AddEquivalentEntity* and *AddSubClass* rewriting rules which can enrich the global ontology without affecting its consistency. The checking of the inconsistencies is done by using the $NACs$.

The *AddEquivalentEntity* rewriting rule adds an equivalent axiom between two entities (two classes or two properties). Figure 7 presents the rewriting rule of the *AddEquivalentClasses* $(C_1, C_2)$ which is defined:

- NACs :
  1. $C_1 \equiv C_2$, condition to avoid redundancy;
  2. $C_1 \sqsubseteq \neg C_2$, two classes cannot be disjoint and equivalent at the same time;
- LHS : $\{C_1, C_2\}$, the classes should exist in the ontology.
- RHS : $(C_1 \equiv C_2)$, the axiom will be added to the ontology.
- DCH : $\emptyset$.

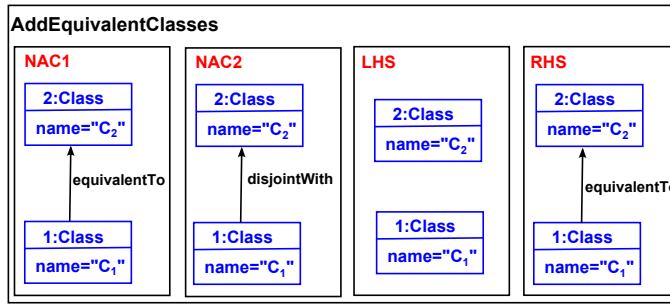



Fig. 7: Rewriting rules for the *AddEquivalentClasses* change.

The *AddSubClass* $(C_1, C_2)$ rewriting rule adds a $subClassOf$ axiom between two classes (Figure 8) and it is defined by:

- NACs :
  1. $C_1 \sqsubseteq C_2$, to avoid redundancy.
  2. $C_2 \sqsubseteq C_1$, the subsumption relation cannot be symmetric;
  3. $C_1 \sqsubseteq \neg C_2$, classes which share a subsumption relation cannot be disjoint;
  4. $\exists C_i \in C(O) \cdot (C_1 \sqsubseteq C_i) \wedge (C_i \sqsubseteq C_2)$, if exist a class $C_i$ which is the $subClassOf$ the class $C_2$ and the $superClass$ of $C_1$, then, $C_1$ is already a $subClass$ of $C_2$;
  5. $\exists (C_i, C_j) \in C(O) \cdot (C_i \sqsubseteq C_1) \wedge (C_j \sqsubseteq C_2) \wedge C_i \sqsubseteq \neg C_j$, classes which share a subsumption relation cannot have subClasses that are disjoint;
- LHS : $\{C_1, C_2\}$, the classes should exist in the ontology.
- RHS : $(C_1 \sqsubseteq C_2)$, the axiom will be added to the ontology.
- DCH : $\emptyset$.

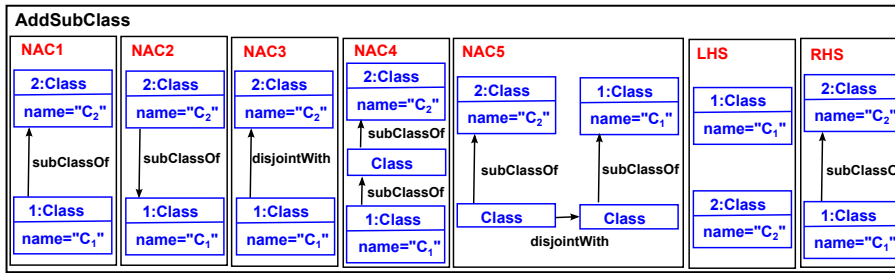



Fig. 8: Rewriting rule for the *AddSubClass* change.

## 4 Implementation and example

In order to implement the proposed method, we have developed a Java program called GROM (Graph Rewriting for Ontology Merging). GROM is based on AGG (Algebraic Graph Grammar) API[5] that supports the algebraic graph transformations (SPO and DPO approaches) and manipulates the typed attributed graph grammars. The tool take as input two ontologies in AGG format (.ggx), a mapping in XML and outputs the merged ontology in AGG format (.ggx). Note that the semantic relations, in the mapping process, are identified by the RitaWN[6] that provides access to the WordNet ontology.

The following presents a merging example of the two ontologies presented in Figure 3. The ontologies were created in OWL using Protégé. They were converted into AGG graphs using the software OWLToGGx [10]. In the following, the word "ontology" is used to refer to its corresponding graph[7].

*Similarity Search* The first step of the proposed approach is the detection of the similarities between the ontologies entities. By considering the ontologies example $O_1$ and $O_2$, the Levenshtein distance returns the following correspondences:

- $CN = \{(\text{“}Tag\text{”}\};\qquad EN = \{(\text{“}hasTag\text{”}, \text{“}has\_Tag\text{”})\}.$

Wordnet detects the synonym terms:

- $SN = \{(\text{“}Emplacement\text{”}, \text{“}Location\text{”})\}.$

The subsumption relations are manually defined:

- $IsaN = \{(\text{“}Company\text{”}, \text{“}Participant\text{”})\}.$

[5] http://user.cs.tu-berlin.de/~gragra/agg

[6] http://rednoise.org/rita/wordnet/documentation

[7] All the materials used in this example (ontologies in OWL and in graph (GGX format) along with the Java implementation) are available for download under open source licence here: http://mariem-mahfoudh.info/medi2014/

*Merging Ontologies* The next step consists in using the set of equivalent nodes $EN$ to replace the nodes of the ontology $O_1$ by their equivalent in the ontology $O_2$. Therefore, the rewriting rule *RenameObjectProperty* is invoked to replace the name of the object properties ($OP$) "hasTag" by "has_Tag". After that, the common ontology graph ($CO$) is created. In our example, it is composed of two nodes ("tag", "hasTag") and an edge which linked them (*range*). To glue the ontologies, the rewriting rule *MergeGraph* is executed. It takes the ontology $O_1$ as a host graph, $CO$ as a $LHS$ and $O_2$ as a $RHS$. Finally, it returns as an output the global ontology ($GO$). Note that all this process if is fully automatic and only takes as parameter the correspondences found in the previous step.

*Global Ontology Adaptation* The global ontology does not yet represent the semantic and subsumption relations. Thus, it is necessary to execute the rewriting rules: 1) *AddEquivalentClasses* ("Emplacement", "Location"); 2) *AddSubClass* ("Company", "Participant"). Figure 9 presents the global ontology resulting of the merging ontologies $O_1$ and $O_2$. Once again, this process is automatic. Finally, the resulting ontology can be easily converted back to OWL[8].

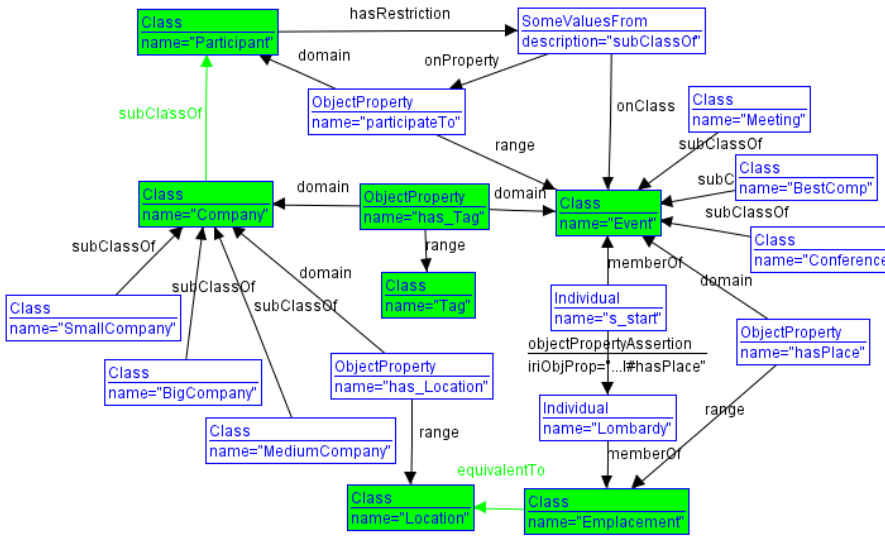



Fig. 9: Result of merging the ontologies $O_1$ & $O_2$.

[8] http://mariem-mahfoudh.info/ksem2013

## 5 Discussion

*Symmetry of the approach* Raunich and al. [22], have presented a state of the art of ontology merging and have distinguished two types of approaches: 1) Symmetric approaches preserve both input ontologies even the redundant data; 2) Asymmetric approaches take one of the ontologies as the source and merge the other as a target. In this type of approach, only the concepts of the source ontology are preserved. Our approach is an asymmetric one with:

$$Merge(O_1, O_2) \neq Merge(O_2, O_1),$$
$$Merge(O_1, O_2) = Merge(O_2, Merge(O_1, O_2) \text{ and}$$
$$Merge(O_2, O_1) = Merge(O_1, Merge(O_2, O_1).$$

However, if the set of equivalent nodes is empty ($EN = \emptyset$) and there is no conflicts between the ontologies, then the "sub-step1" of the ontologies merging is not executed and the approach, in this case, is symmetric. Thus, we have:

$$Merge(O_1, O_2) = Merge(O_2, O_1) = Merge(O_1, Merge(O_1, O_2)) = Merge(O_2, Merge(O_1, O_2)) = GO.$$

*Coverage* The coverage ($Cov$) is a criteria to evaluate the quality of merge results. It is is related to the degree of information preservation and measures the share of input concepts preserved in the result. Coverage values is between 0 and 1 [22]. In our approach, we distinguish two cases:

1. if $EN = \emptyset$, then, $Cov = 1$. All the ontologies concepts are preserved but with the advantage that the redundant data (*e.g.* the multiple inheritance) are dropped (thanks to the application of the $NACs$ in the rewriting rules).
2. if $EN \neq \emptyset$, then, $Cov = 1 - \frac{card(EN(O_1))}{card(N(O_1)+N(O_2))}$. Only the equivalent nodes of the ontology source ($O_1$) are lost.

*Complexity* The most demanding step in time and resource is the recognition of the $LHS$ from the host graph $G$. This research is an NP-complete problem. More precisely, a search of a sub-graph composed of $k$ elements in a graph compound of $n$ elements has a complexity of $O(n^k)$. However, the cost of calculation remains quite acceptable if the size of the $LHS$ graph is limited [23]. In the most examples of transformations, this condition is satisfied.

*Conflicts management* In this article, ontologies are expected to be correct and there is no strong contradictions between them. However, as they conceptualize identical or related domains, ontologies may have some conflicts. Therefore, another step should be added to detect the possible conflicts. Thus, the addition of axioms into the target ontology should be sequential and the user intervention may be required. Several cases can be found. We present in this section how to add individuals axioms of the target ontology without affecting the source ontology. In particular, the rewriting rule $AddObjectPropertyAssertion(I_1, I_2, OP)$ is discussed. It adds an $ObjectPropertyAssertion$ between two individuals and it is defined as follow:

– NAC :
  1. $(I_1, I_2) \in OP$, to avoid redundancy.
  2. $\exists I_i \in I(O) \cdot (I_i \neq I_2) \wedge ((I_1, I_i) \in OP)$[9] $\wedge (\top \sqsubseteq 1OP)$, if there is an individual $I_i$ which different to $I_2$ and it is linked to $I_1$ by $OP$, where $OP$ is a functional property, then the addition of the assertion is not allowed;
  3. $\exists \leqslant nOP \cdot (\exists I_i \in I(O)) \wedge \{(I_1, I_i) \in OP\} = n$, if there is a restriction ($OP$ maxCardinality(n)) and the count of the assertion individuals is equal to n, then, the adding of other assertion is not allowed.
– LHS : $\{I_1, I_2, OP\}$, $I_1$, $I_2$ and $OP$ should exist in the ontology.
– RHS : $((I_1, I_2) \in OP)$, the assertion should be added to the ontology.
– DCH : $\emptyset$.

## 6 Related work

A limited number of approaches were proposed for merging ontologies. They can be classified into two main categories: the approaches based on semantic web technologies [24,25,26,27,28] and the approaches based on algebraic specification and Category Theory [29,5,7].

The aim of our work is to present a formal approach for ontologies merging and show that typed graph grammars can be a good formalism to manage ontology changes (evolution and merging). Therefore, we have studied the existing propositions in the domain of algebraic specifications. Thus, Zimmermann et al., [5] have presented a categorical approach to formalize ontologies alignment. They proposed two formalisms: the V-alignment and W-alignment which use the "span" concept of the Category Theory. This work is an important reference as it presents the foundations of using categories in the field of semantic web. However, it is focused on the problem of ontologies alignment and it studied the merge only as an operation of alignment. Later, Cafezeiro et al., [7] have proposed to use the concepts of Category Theory ("limit", "colimit" and "pushout") to formalize the ontology operations. This work defines merge and composition operations but only considers ontologies which are composed of classes, hierarchies of classes and relations. It does not consider neither the individuals nor the axioms. Finally, these approaches have not been implemented. In our work, we use the algebraic approach and category theory in the frame of graph grammars formalism. This allowed us to implement the proposed approach (GROM) and to benefit the application conditions (e.g. NACs) to avoid inconsistencies. Furthermore, our approach is more general because it treats individuals and axioms.

## 7 Conclusion

In this paper, we presented a formal approach for merging ontologies using typed graph grammars. It is divided into three steps. The first searches correspondences

[9] $I_1$ and $I_i$ are linked by the objectProperty $OP$.

between nodes from the ontologies. It is based on lexical techniques (Levenshtein distance) and an external resource (the linguistic ontology WordNet). The second step merges the structures of the ontologies using the correspondences computed in the previous step, by using the SPO approach. The last step, enriches the merged ontology with subsumption and semantic relations. It used for that the rewriting rules of some basic ontology changes (*AddEquivalentEntity*, *AddSubClass*, etc.). To validate our proposals, we have implemented a new tool, GROM that given two ontologies and their mapping, it is able to generate the global ontology automatically. As it is based on the the algebraic graph transformations, it allows to define a simple and formal way to merge ontologies while encapsulating the complex details of their structures.

For future work, we intend to study the different conflicts which can affect the result of the merge. Then, we plan to improve the alignment result and explore other techniques of similarity search specially the structural techniques. The current test case study includes small ontologies, we are currently considering larger ontologies in order to perform a better evaluation of the method.

## Acknowledgment

The authors would like to thank the European project CCAlps which funded this work (project number is 15-3-1-IT).

## References

1. Staab, S., Studer, R.: Handbook on ontologies. Springer (2010)
2. Rosse, C., Mejino Jr, J.L.: A reference ontology for biomedical informatics: the foundational model of anatomy. Journal of biomedical informatics **36**(6) (2003) 478–500
3. Klein, M.: Combining and relating ontologies: an analysis of problems and solutions. In: IJCAI-2001, Workshop on ontologies and information sharing. (2001) 53–62
4. Ehrig, H., Montanari, U., Rozenberg, G., Schneider, H.J.: Graph Transformations in Computer Science. Geschäftsstelle Schloss Dagstuhl (1996)
5. Zimmermann, A., Krotzsch, M., Euzenat, J., Hitzler, P.: Formalizing ontology alignment and its operations with category theory. Frontiers in Artificial Intelligence and Applications **150** (2006) 277– 288
6. d'Aquin, M., Doran, P., Motta, E., Tamma, V.A.: Towards a parametric ontology modularization framework based on graph transformation. In: WoMO. (2007)
7. Cafezeiro, I., Haeusler, E.H.: Semantic interoperability via category theory. In: 26th international conference on Conceptual modeling, Australian Computer Society, Inc. (2007) 197–202
8. De Leenheer, P., Mens, T.: Using graph transformation to support collaborative ontology evolution. In: Applications of Graph Transformations with Industrial Relevance. Springer (2008) 44–58
9. Javed, M., Abgaz, Y.M., Pahl, C.: Ontology change management and identification of change patterns. Journal on Data Semantics **2**(2-3) (2013) 119–143

10. Mahfoudh, M., Forestier, G., Thiry, L., Hassenforder, M.: Consistent ontologies evolution using graph grammars. In: Knowledge Science, Engineering and Management (KSEM). Springer (2013) 64–75
11. Rozenberg, G.: Handbook of graph grammars and computing by graph transformation. Volume 1. World Scientific (1999)
12. Ehrig, H., Pfender, M., Schneider, H.J.: Graph-grammars: An algebraic approach. In: Switching and Automata Theory (SWAT, IEEE (1973) 167–180
13. Barr, M., Wells, C.: Category theory for computing science. Volume 10. Prentice Hall New York (1990)
14. Fokkinga, M.M.: A gentle introduction to category theory — the calculational approach. In: Lecture Notes of the STOP 1992 Summerschool on Constructive Algorithmics. University of Utrecht (1992) 1–72
15. Löwe, M.: Algebraic approach to single-pushout graph transformation. Theoretical Computer Science **109**(1) (1993) 181–224
16. Ehrig, H.: Introduction to the algebraic theory of graph grammars (a survey). In: Graph-Grammars and Their Application to Computer Science and Biology, Springer (1979) 1–69
17. Ivanov, P., Voigt, K.: Schema, ontology and metamodel matching - different, but indeed the same? In: First International Conference on Model and Data Engineering (MEDI), Springer (2011) 18–30
18. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions and reversals. In: Soviet physics doklady. Volume 10. (1966) 707–710
19. Do, H.H., Rahm, E.: Coma: a system for flexible combination of schema matching approaches. In: 28th international conference on Very Large Data Bases (VLDB), VLDB Endowment (2002) 610–621
20. Miller, G.A.: Wordnet: A lexical database for english. Communications of the ACM **38**(11) (1995) 39–41
21. Shvaiko, P., Euzenat, J.: Ontology matching: state of the art and future challenges. (2012)
22. Raunich, S., Rahm, E.: Towards a benchmark for ontology merging. In: Move to Meaningful Internet Systems: OTM 2012 Workshops, Springer (2012) 124–133
23. Karsai, G., Agrawal, A., Shi, F., Sprinkle, J.: On the use of graph transformations in the formal specification of computer-based systems. In: IEEE TC-ECBS and IFIP10. 1 Joint Workshop on Formal Specifications of Computer-Based Systems. (2003) 19–27
24. Noy, N.F., Musen, M.A.: Algorithm and tool for automated ontology merging and alignment. In: 17th National Conference on Artificial Intelligence (AAAI), AAAI Press/The MIT Press (2000) 450–455
25. Nováček, V., Smrz, P.: Empirical merging of ontologies-a proposal of universal uncertainty representation framework. In: The 3rd Annual European Semantic Web Conference (ESWC), Springer (2006) 65–79
26. Li, G., Luo, Z., Shao, J.: Multi-mapping based ontology merging system design. In: 2nd International Conference onAdvanced Computer Control (ICACC). Volume 2., IEEE (2010) 5–11
27. Raunich, S., Rahm, E.: Atom: Automatic target-driven ontology merging. In: 27th International Conference on Data Engineering (ICDE), IEEE (2011) 1276–1279
28. Fareh, M., Boussaid, O., Chalal, R., Mezzi, M., Nadji, K.: Merging ontology by semantic enrichment and combining similarity measures. International Journal of Metadata, Semantics and Ontologies **8**(1) (2013) 65–74
29. Hitzler, P., Krötzsch, M., Ehrig, M., Sure, Y.: What is ontology merging? In: American Association for Artificial Intelligence. (2005)