What to Do When the Users of an Ontology Merging System Want the Impossible? Towards Determining Compatibility of Generic Merge Requirements

Samira Babalou, Elena Grygorova, Birgitta König-Ries

Heinz-Nixdorf Chair for Distributed Information Systems
Institute for Computer Science, Friedrich Schiller University Jena, Germany
{samira.babalou,elena.grygorova,birgitta.koenig-ries}@uni-jena.de

Abstract. Ontology merging systems enable the reusability and interoperability of existing knowledge. Ideally, they allow their users to specify which characteristics the merged ontology should have. In prior work, we have identified Generic Merge Requirements (GMRs) reflecting such characteristics. However, not all of them can be met simultaneously. Thus, if a system allows users to select which GMRs should be met, it needs a way to deal with incompatible GMRs. In this paper, we analyze in detail which GMRs are (in-)compatible, and propose a graph based approach to determining and ranking maximum compatible supersets of user-specified GMRs. Our analysis shows that this is indeed feasible to detect the compatible supersets of the given GMRs that can be fulfilled simultaneously. This approach is implemented in the open source $\mathcal{C}o\mathcal{M}erger$ tool.

Keywords: Ontology merging . Merge requirements . Graph theory

1 Introduction

An ontology is a formal, explicit description of a given domain. It contains a set of entities, including classes, properties, and instances. Ontology merging [1] is the process of creating a merged ontology $\mathcal{O}_M$ from a set of source ontologies $\mathcal{O}_S$ with a set of corresponding pairs extracted from a given mapping. Various ontology merging systems [2–16] provide different sets of criteria and requirements that their merged ontologies should meet. In [17], we have analyzed the literature and determined which criteria, called Generic Merge Requirements (GMRs), are used by different approaches. Customizing the GMRs within an ontology merging system provides a flexible merging approach, where users can actively choose which requirements are important to them, instead of allowing only a very indirect choice by picking a merge system that uses their preferred set of criteria. Unfortunately, not all GMRs are compatible. For instance, one may want to preserve all properties contained in the original ontology in the merged

ontology. On the other hand, one could wish to avoid cycles. Likely, these goals conflict.

The motivation behind this work is to enable merging systems to take user input into consideration, so ultimately, to have user-requirement driven ontology merging. Our proposal allows flexibility on the user side to select an arbitrary set of GMRs. Thus, once a user has chosen a set of important GMRs, a system is needed to check their compatibility and suggest a maximum set of requirements that can be met simultaneously. In this paper, we analyze in detail the (in)compatibility of GMRs and describe a graph based approach to determining maximal compatible sets for the given GMRs. Further, an automatic ranking method is proposed on the set of the system suggested compatible sets. The proposed framework is conservative and finds potential conflicts in GMRs. For a given ontology, not all of these potential conflicts may materialize. We discuss in Sec. 3, how the approach could be extended to leverage this. GMRs are implemented in $\mathcal{CoMerger}$ [18] and are publicly available and distributed under an open-source license. We have empirically analyzed various merged ontologies for the given set of user-selected GMRs, and observed that there is a superset of compatible GMRs that can be fulfilled simultaneously.

The rest of the paper is organized as follows. Sec. 2 surveys GMRs. The proposed method of compatibility checker of GMRs is presented in Sec. 3. In Sec. 4, the compatible sets are ranked. An empirical analysis of GMRs is demonstrated in Sec. 5. The paper is concluded in Sec. 6.

2 Survey on Generic Merge Requirements

Generic Merge Requirements (GMRs) are a set of requirements that the merged ontology is expected to achieve. GMRs have been first introduced in the Vanilla system [1]. Later other merge approaches implicitly or explicitly took them into consideration [2–8, 10–16, 19, 20]. To provide customizable GMRs in an ontology merging system, we surveyed the literature to compile a list of GMRs. This investigation lead to twenty GMRs [17], summarized in Table 1. They have been categorized in six aspects: completeness (*R1-R7*), minimality (*R8-R11*), deduction (R12), constraint (*R13-R15*), acyclicity (*R16-R18*), and connectivity (*R19* and *R20*). This list has been acquired by studying three different research fields:

1. *Ontology and model merging methods*: The GMRs *R1-R6*, *R8-R16*, *R19* have been extracted from existing ontology and model merging methods such as [2, 3, 6]. These approaches aim to implicitly or explicitly meet at least one of the GMRs on their merged ontology.
2. *Ontology merging benchmarks*: The existing benchmarks [21, 22] on the ontology merging domain introduced general desiderata and essential requirements that the merged ontology should meet. The criteria stated in these benchmarks are based on earlier research in [23], a study of the quality measurement of the merged ontology. In this respect, *R1*, *R4*, *R7- R9* have been extracted from these research studies.

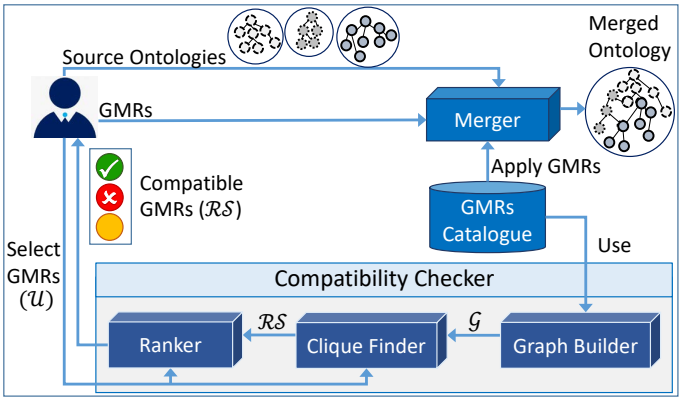

Fig. 1: The GMRs' compatibility checker within the ontology merge system.

3. *Ontology engineering*: Researchers of the ontology engineering domain [24–26] came up with a set of criteria to present the correctness of an ontology, which is developed in a single environment. It is worthwhile to consider these criteria also on the merged ontology because the newly created merged ontology may be viewed the same as the developed ontologies in this category. Not all of these criteria can be extended in the ontology merging scenario since some relate to the problem of the source ontologies modeling, in which the merge process can not affect them. In this regards, we recast *R15* - *R20* from this category.

To the best of our knowledge, there is no general compatibility checker between the GMRs in the literature. However, the approach in [27] proposed a resolution for conflicts that occurred by applying *R13* and *R14*.

3 Proposed Approach for Checking GMRs Compatibility

In this section, we describe our approach to finding maximum compatible supersets of user-specified GMR. Basically, what we do is first find subsets of the GMRs specified by the user that are compatible, and second, extend those by further GMRs, (out of those the user had not selected), while maintaining compatibility. Our intuition is that first, as much as possible of what the user wanted should be met and that second, adding further GMRs will, in general, improve the quality of the merged ontology.

Therefore, a framework is required to detect the compatibility of the user-selected GMRs. We propose such a framework within the ontology merge system, as shown in Fig. 1. The source ontologies are merged based on the user-selected GMRs. Users can request to check the compatibility of the selected GMRs. To achieve this, we build a graph $\mathcal{G}$ of the interactions between the GMRs. We then recast the problem at hand by selecting the maximal superset

Table 1: Generic Merge Requirements (GMRs).

***R1*- Class preservation**: All classes of source ontologies should be preserved in the merged ontology [1, 2, 4–11, 21–23].
***R2*- Property preservation**: All properties of source ontologies should be preserved in the merged ontology [1, 2, 4, 5, 12].
***R3*- Instance preservation**: All instances of source ontologies should be preserved in the merged ontology [1, 2, 4, 13].
***R4*- Correspondence preservation**: The corresponding entities from source ontologies should be mapped to the same merged entity [1, 2, 8, 22].
***R5*- Correspondences' property preservation**: The merged entity should have the same property of its corresponding entities [1, 8].
***R6*- Property's value preservation**: Properties' values from the source ontologies should be preserved in the merged ontology [1, 8].
***R7*- Structure preservation**: The hierarchical structure of source ontologies' entities should be preserved in the merged ontology [23].
***R8*- Class redundancy prohibition**: No redundant classes should exist in the merged ontology [1, 2, 5, 10, 14–16, 21, 23].
***R9*- Property redundancy prohibition**: No redundant properties should exist in the merged ontology [4, 12, 21].
***R10*- Instance redundancy prohibition**: No redundant instances should exist in the merged ontology [7, 13].
***R11*- Extraneous entity prohibition**: No additional entities other than the source ontologies' entities should be added in the merged result [1].
***R12*- Entailment deduction satisfaction**: All entailments of the source ontologies should be satisfied in the merged ontology [3, 5].
***R13*- One type restriction**: Any merged entity should have one data type [1].
***R14*- Property value's constraint**: Restriction on property's values from source ontologies should be applied without conflict in the merged ontology [1, 3].
***R15*- Property's domain and range oneness**: The merge process should not result in multiple domains or ranges defined for a single property [25].
***R16*- Acyclicity in the class hierarchy**: The merge process should not produce a cycle in the class hierarchy [1, 2, 4, 6, 11, 19, 23–25].
***R17*- Acyclicity in the property hierarchy**: The merge process should not produce a cycle between properties w.r.t. the is-subproperty-of relationship [20, 25].
***R18*- Prohibition of properties being inverses of themselves**: The merge process should not cause an inverse recursive definition on the properties [25].
***R19*- Unconnected class prohibition**: Each connected class from source ontologies should not be unconnected in the merged ontology [1, 6, 25].
***R20*- Unconnected property prohibition**: Each connected property from the source ontologies should not be unconnected in the merged ontology [8, 25].

of the user-selected GMRs as $\mathcal{RS} = \{rs_1, rs_2, ..., rs_z\}$. These results are ranked, sorted, and returned to the user. More precisely, the framework performs the following steps:

1. A graph $\mathcal{G}$ is built based on the interactions between GMRs.

Table 2: Scope of changes by applying GMRs in the merged ontology.

Scope	Sub-Scope	Explanation
Scope 1- Classes	Scope 1_1	Classes origin from source ontologies
	Scope 1_2	Redundant classes
	Scope 1_3	Extra classes that do not belong to any source ontologies
Scope 2 Properties	Scope 2_1	Properties origin from source ontologies
	Scope 2_2	Redundant properties
	Scope 2_3	Extra properties that do not belong to any source ontologies
Scope 3- Instances	Scope 3_1	Instances origin from source ontologies
	Scope 3_2	Redundant instances
	Scope 3_3	Extra instances that do not belong to any source ontologies
Scope4- Values of properties	-	Values of properties in the merged ontology

2. The compatible subsets of the user-selected GMRs are extracted from the $\mathcal{G}$. Then, they will be extended to the maximal compatible superset.
3. The detected sets are ranked and ordered.
4. An ordered list of compatible sets is returned to the user.

In the next sub-section, building the GMRs interaction graph $\mathcal{G}$ and extracting the compatible supersets in the graph will be explained.

3.1 Building GMRs Interactions Graph $\mathcal{G}$

The *Graph Builder* component in Fig. 1 takes as input the GMRs catalogue and creates the graph $\mathcal{G}$. The GMRs' interaction graph $\mathcal{G} = (V, E)$ demonstrates the interaction between GMRs, where V is the set of vertices representing the GMRs (see Table 1), and E is the set of edges. In this graph, two GMRs are connected via an edge if they are compatible. To define the compatibility of GMRs (existence of an edge between two GMRs in $\mathcal{G}$), two conditions are defined:

Condition I. The scope of changes by a GMR on the merged ontology can reveal its (in-)compatibility with others. Thus, two GMRs are compatible if they do not modify the same scope of entities. The changes made by each GMR are applied to the classes (scope 1), properties (scope 2), instances (scope 3), and value of properties (scope 4), defined by the union of sub-scopes, as shown in Table 2. We distinguish between two scopes:

- **Direct scope**: It is the main scope that is affected by applying a GMR. E.g., applying *R1* adds missing classes, so the direct scope of *R1* is the classes.
- **Indirect scope**: It is the scope that might be affected by the changes made on the direct scope. E.g., applying *R8* deletes the redundant classes (direct scope is redundant classes). However, as a side effect of this operation, this might cause the properties connected to those classes to become unconnected, or their instance to be orphaned. Thus, the indirect scopes of *R8* are properties and instances.

Condition II. Let us illustrate our intuition to require the second condition with an example by considering *R2* (property preservation) and *R5* (correspondences' property preservation). *R2* may make changes to the properties, and *R5* possibly makes changes to the properties of the corresponding classes. So, both GMRs apply changes on the same set of entities (Scope 2_1). However, it cannot be concluded that both GMRs are incompatible because the operations that both carry on the merged ontology do not have any contradiction. *R2* uses the add operation to preserve the missing properties. *R5* also uses the add operation to add missing properties of the corresponding classes. So, both these actions can be performed simultaneously in the merged ontologies without conflict. As a whole, three types of operation are performed to meet the GMRs and ensure their fulfillment: (1) Add: e.g., *R1* uses the add operation to preserve the missing classes in the merged ontology. (2) Delete: e.g., *R8* uses the delete operation to get rid of redundant classes. (3) A combination of add and delete: e.g., *R4* uses add and delete operations, in which for two corresponding classes c_1 and c_2 that are not mapped to the integrated class c', first, c_1 and c_2 will be deleted, then c' will be added. Table 3 shows the scopes and operations of each GMR. For some GMRs, there are different possible operations. We followed one solution in this paper and marked the alternative one by the symbol *.

Although applying each GMR may change direct and indirect scopes, their operations carry on the direct scope. Therefore, to determine the compatibility of the GMRs, the type of operations performed by each GMR on the direct scope should be taken into account. In this regards, when two GMRs change the same set of entities, they can still be compatible if both use the same operation. Let $\mu(R_j)$ be a set of entities that get affected by applying $R_j \in$ GMRs on the merged ontology, i.e., the direct scope. Given the conditions mentioned above, we define the compatibility between R_j and $R_k \in$ GMRs as:

Definition 1. *R_j is compatible with R_k ($R_j \parallel R_k$) if R_j and R_k modify different scopes of entities in the merged ontology, i.e., $\mu(R_j) \neq \mu(R_k)$. If $\mu(R_j) = \mu(R_k)$, the type of operation of the applying R_j and R_k should be the same.*

Accordingly, there could be four variants between the scope of changes and the types of operation, as:

***Case A*- Same Scopes and Same Operations**: In this case, the scope of entities affected by applying R_j and R_k, is the same. Moreover, R_j and R_k use the same type of operations. Since both GMRs use the same operation on the same set of entities, they are compatible with each other.

- *Example*: *R2* $\parallel$ *R7*. *R2* and *R7* both make changes in the properties origin from the source ontologies. Moreover, *R2* uses the add operation to add the missing properties. *R7* uses the add operation to add the missing is-a properties in order to preserve the hierarchy structure of the source ontologies in the merged ontology. Thus, both are compatible.

***Case B*- Same Scopes with Different Operations**: In this case, the set of entities, getting affected by applying R_j and R_k, is the same. However, R_j and R_k

Table 3: The scopes and operations of each GMR. The symbol * indicates an alternative solution.

GMR	Direct Scope	Indirect Scope	Operation	Description
R1	$S1_1$	-	add	It adds missing classes of the source ontologies.
R2	$S2_1$	-	add	It adds missing properties of the source ontologies.
R3	$S3_1$	-	add	It adds missing instances of the source ontologies.
R4	$S1_1$	$S2$ $S3$	add & delete	If two corresponding classes c_1 and c_2 are not mapped to the one integrated class c', first, c_1 and c_2 is deleted, then c' will be added.
	$S2_1$	$S1$ $S3$	add & delete	It follows the procedure the same as the *R4*-scope 1-1 but one the properties.
R5	$S2_1$	-	add	It adds missing properties of the corresponding classes.
R6	$S4$	-	add	It adds missing values of the properties.
R7	$S2_1$	-	add	It adds is-a properties to the respective class.
R8	$S1_2$	$S2, S3$	delete	It deletes redundant classes.
R9	$S2_2$	$S1, S3$	delete	It deletes redundant properties.
R10	$S3_2$	$S1$	delete	It deletes redundant instances.
R11	$S1_3$ $S2_3$ $S3_3$	$S1, S2,$ $S3$	delete	It deletes extra entities.
R12	$S1_1$ $S2_1$ $S3_1$	- - -	add	It adds some entities to achieve the entailment the same as the source ontologies.
R13	$S4$	-	delete	It keeps only one of the data types and deletes the other one.
R14	$S4$	-	delete	It keeps only one value of the property and deletes the other one.
R15	$S1_1$	$S2, S3$	add & delete	It might add multiple domains or ranges as the unionOf to the property.
			delete*	It might delete multiple domains or ranges and only keep one of them.
R16	$S2$	$S1$	delete	It might delete some properties to be free of cycles.
	$S1$	$S2, S3$	delete*	It might delete some classes to be free of cycles.
R17	$S2$	$S1$	delete	It deletes properties to be free of the cycle on the properties' hierarchy.
R18	$S2$	$S1$	delete	It deletes the inverse of properties.
R19	$S2$	-	add	It might add is-a relations to connect the unconnected classes.
	$S1$	$S2, S3$	delete*	It might delete unconnected classes.
R20	$S2$	$S1$	delete*	It might delete the unconnected properties.
		-	add	It might use the add operation to connect the unconnected properties to the classes.

use different types of operations. Since both use the different operations on the same set of entities, they are incompatible with each other.

- *Example*: *R2* $\nparallel$ *R17*. Both *R2* and *R17* change properties. *R2* adds missing properties, whereas *R17* may delete some properties to achieve acyclicity. Thus, it may happen that applying *R17* reverses the changes made by *R7* and vice versa.

Case C- **Different Scopes with the Same Operations**: In this case, the set of entities, getting effect by applying R_j and R_k, is different. Moreover, both use the same type of operations. Since both GMRs using the same operation but on different sets of entities, they are compatible.

- *Example*: *R1* $\parallel$ *R2*. Preserving the classes in the merged ontology will make some changes in the classes in *R1*. However, preserving the properties will modify the properties in *R2*. These two GMRs do not change the same group of entities. Moreover, both use the add operation for applying these GMRs. Since both GMRs use the same operation but on different sets of entities, they are compatible.

Table 4: Compatibility interaction between GMRs. f_d shows the compatibility degree.

GMR	Compatible GMRs	f_d
R1	*R2*, *R3*, *R5-R14*, *R16-R20*	0.89
R2	*R1*, *R3*, *R5-R15*, *R19*, *R20*	0.79
R3	*R1*, *R2*, *R4-R20*	1
R4	*R3*, *R6*, *R8-R11*, *R13*, *R14*	0.74
R5	*R1-R3*, *R6-R15*, *R19*, *R20*	0.79
R6	*R1-R5*, *R7-R12*, *R15-R20*	0.89
R7	*R1-R3*, *R5*, *R6*, *R8-R15*, *R19*, *R20*	0.79
R8	*R1-R7*, *R9-R20*	1
R9	*R1-R8*, *R10-R18*	0.89
R10	*R1-R9*, *R11-R20*	1
R11	*R1-R10*, *R12-R18*	0.89
R12	*R1-R3*, *R5-R11*, *R13*, *R14*, *R19*, *R20*	0.74
R13	*R1-R5*, *R7-R12*, *R14-R20*	0.95
R14	*R1-R5*, *R7-R13*, *R15-R20*	0.95
R15	*R2*, *R3*, *R5-R11*, *R13*, *R14*, *R16-R20*	0.84
R16	*R1*, *R3*, *R6*, *R8-R11*, *R13-R15*, *R17*, *R18*	0.63
R17	*R1*, *R3*, *R6*, *R8-R11*, *R13-R16*, *R18*	0.63
R18	*R1*, *R3*, *R6*, *R8-R11*, *R13-R17*	0.63
R19	*R1-R3*, *R5-R8*, *R10*, *R12-R15*, *R20*	0.68
R20	*R1-R3*, *R5-R8*, *R10*, *R12-R15*, *R19*	0.68

Case D- **Different Scopes and Different Operations**: In this case, applying R_j and R_k is performed on different sets of entities. Moreover, both use different types of operations. Since both use different operations on the different entity sets, they are completely separated and do not have any effect on each other. So, they are compatible.

- *Example*: *R1* $\parallel$ *R11*. *R1* makes changes in the classes origin from source ontologies (scope 1_1). *R11*, in addition to changing properties, modifies the extra classes (scope 1_3). So, the scopes of changes by these two GMRs are on the different entity sets. *R1* uses the add operation, while *R11* uses delete operation. Since both use the different operations on different sets of entities, they are compatible.

Considering the scope and the operation of each GMR, the interaction between GMRs can be concluded in Table 4, in which R_j is considered compatible with R_k if the intersection of all its sub-scopes is compatible. Thus, the graph $\mathcal{G}$ has edges between the compatible GMRs, as stated in Table 4. The compatibility degree f_d for each GMR R_j is the number of compatible GMRs with R_j divided by the total number of GMRs, as shown in the last column. *R3*, *R8*, and *R10* are compatible with all other GMRs. *R13* and *R14* have high compatibility as the scope of their changes is different from the others. *R16*, *R17*, and *R18* are the least compatible.

3.2 Clique Finder

Given the GMRs interaction graph $\mathcal{G}$ and the set $\mathcal{U}$ containing the GMRs the user is interested in, we aim to find the maximal superset of V containing all vertices out of $\mathcal{U}$ and no incompatible nodes. This may not always be achievable since the user might have chosen incompatible GMRs already. In this case, we search for a maximal superset of V in $\mathcal{G}$ that preserves as many nodes out of $\mathcal{U}$ as possible and contains compatible nodes only. Thus, the *Clique Finder* component in Fig. 1 takes as input a set of user-selected GMRs $\mathcal{U}$ alongside with the GMRs' interaction graph $\mathcal{G}$. It returns a set of all possible compatible sets, namely $\mathcal{RS} = \{rs_1, rs_2, ..., rs_l\}$. Each suggested compatible set $rs \in \mathcal{RS}$ contains (all/part) of the user-selected compatible GMRs, and compatible GMRs additionally all other. For the given user-selected GMRs, each suggested compatible set rs is formulated in Eq. 1.

$$rs = \mathcal{U}^C \cup \mathcal{U}^{EC} \tag{1}$$

where, $\mathcal{U}^C$ is a compatible subset of $\mathcal{U}$, and $\mathcal{U}^{EC}$ is an extra compatible set of GMRs related to $\mathcal{U}$. To obtain the compatible set rs, we recast the problem at hand as clique extraction on the GMRs' interaction graph $\mathcal{G}$, where it needs to be the maximal best match based on the user-selected GMRs. A clique is a set of fully connected vertices. Thus a compatible clique $\mathcal{K}^C$-*Clique* is extracted, where $\mathcal{K}$ indicates the number of vertices in the clique, and C denotes that the clique is compatible.

Definition 2. *The $\mathcal{K}^C$-Clique is a compatible clique iff between all vertices only the compatible relations exist.*

Compatible relations between GMRs are encoded by edges in the GMRs interaction graph $\mathcal{G}$. Thus, $\mathcal{K}^C$-*Clique* includes compatible GMRs from $\mathcal{U}$ (called $\mathcal{U}^C$) and additional compatible GMRs related to $\mathcal{U}$'s elements (called $\mathcal{U}^{EC}$). $\mathcal{K}^C$-*max-Clique* is a clique containing at least $\mathcal{K}$ vertices that is not a subset of any other cliques. To compute the $\mathcal{K}^C$-*max-Clique*, we use the CLIQUES algorithm described in [28]. To avoid enumerating all possible subgraphs, two constraints on the clique extraction are placed:

1. If a clique does not contain at least $\mathcal{K}$ vertices, then neither the clique nor any other sub-cliques can contain a $\mathcal{K}^C$-*Clique*, because, if the clique does not have the required number of vertices, it cannot be a $\mathcal{K}^C$-*Clique*.
2. Only vertices in a $\mathcal{K}^C$-*max-Clique* of $\mathcal{G}$ can form a $\mathcal{K}^C$-*Clique*, because a vertex that is not in a $\mathcal{K}^C$-*max-Clique* cannot be in any $\mathcal{K}^C$-*Clique*.

The first constraint contributes to reducing the search space, and the second one narrows the result to the maximal desired compatible GMRs. Moreover, Definition 2 guarantees that the selected GMRs are compatible.

4 Ranking the Compatible Sets

For each set of user-selected GMRs, there are different possible compatible GMRs sets. Let $\mathcal{RS} = \{rs_1, rs_2, ..., rs_l\}$ be all possible compatible sets based on the

user-selected GMRs. To figure out which $rs_z \in \mathcal{RS}$ is the best choice, the *Ranker* component in Fig. 1 rates the elements of $\mathcal{RS}$ based on different criteria. The ranked values assign a confidence degree to each suggested compatible set.

Assume that the user selected $\mathcal{U} = \{R7, R9, R10, R16\}$. The approach described before finds three possible compatible sets, $\mathcal{RS} = \{rs_1, rs_2, rs_3\}$[1], where $rs_1 = \{R1, R3, R6, R8, R9, R10, R11, R16, R17, R18\}$, $rs_2 = \{R1, R3, R8, R9, R10, R11, R13, R14, R16, R17, R18\}$, and $rs_3 = \{R2, R3, R5, R6, R7, R8, R9, R10, R11, R15\}$. To determine which rs is the best choice, we rank all compatible sets with three different criteria:

1. **The number of user-selected GMRs in each compatible set:** The intersection of the compatible set rs_z and the user-selected GMRs $\mathcal{U}$, $rs_z \cap \mathcal{U}$, comprises all elements which are contained in both rs_z and $\mathcal{U}$. Therefore, we count the number of elements that are available in both rs_z and $\mathcal{U}$. Let us consider that $|rs_z|$ is the number of GMRs in the compatible set rs_z, $|\mathcal{U}|$ is the number of GMRs in the user-selected GMRs ($\mathcal{U}$), $|\mathcal{U} \cap rs_z|$ is the number of GMRs contained in both rs_z and $\mathcal{U}$, and $|GMRs|$ the total number of GMRs in our system. Given these notations, Eq. 2 ranks each suggested compatible set based on considering the user preference in the first part and the power (important) of rs_z itself in the second part.

$$Score_1(rs_z) = \frac{|\mathcal{U} \cap rs_z|}{|\mathcal{U}|} + \frac{|rs_z|}{|GMRs|} \tag{2}$$

   In the given example, rs_1 has $|rs_1| = 10$, $|\mathcal{U}| = 4$, $|\mathcal{U} \cap rs_1| = 3$, and $|GMRs| = 20$. Therefore, this score is obtained as $Score_1(rs_1) = \frac{3}{4} + \frac{10}{20} = 1.25$. Similarly, this score for rs_2 and rs_3 is calculated as $Score_1(rs_2) = 1.3$ and $Score_1(rs_3) = 1.25$.
2. **The number of user-selected aspects in each compatible set:** GMRs have been categorized in different aspects, which users can select. So, not only the number of user-selected GMRs has an effect on the ranking of each compatible set rs, also the user-selected aspects should be taken into account. Therefore, we calculate to which extent each suggested compatible set rs_z covers the user's intended aspects. Let us $\Psi(\mathcal{U})$ be the number of GMRs' aspects in $\mathcal{U}$, $\Psi(rs_z)$ the number of GMRs' aspects in rs_z, $\Psi(\mathcal{U} \cap rs_z)$ the number of common aspects in both rs_z and $\mathcal{U}$, and $|GMRs_{Aspect}|$ the total number of aspects in the GMRs catalouge. Given these notations, Eq. 3 ranks each suggested compatible set based on considering the user preference aspects in the first part and the power (important) of rs_z's aspect itself in the second part of the equation.

$$Score_2(rs_z) = \frac{\Psi(\mathcal{U} \cap rs_z)}{\Psi(\mathcal{U})} + \frac{\Psi(rs_z)}{|GMRs_{Aspect}|} \tag{3}$$

   In the current example, rs_1 has $\Psi(\mathcal{U}) = 3$, $\Psi(rs_1) = 3$, $\Psi(\mathcal{U} \cap rs_1) = 3$, and $|GMRs_{Aspect}| = 6$. Thus, this score is obtained as $Score_2(rs_1) = \frac{3}{3} + \frac{3}{6} = 1.5$.

[1] For the given $\mathcal{U}$, there are 18 different maximal compatible sets. To make the example concise, we consider 3 compatible sets, only.

Fig. 2: Top-10 maximum compatible sets for $\mathcal{U}$ ={*R7*, *R9*, *R10*, *R16*}.

Similarly, this score for rs_2 and rs_3 is achieved as $Score_2(rs_2) = 1.67$ and $Score_2(rs_3) = 1.5$.

3. **Compatibility degree of each GMR:** Up to now, the proposed metrics consider the quantity measure. This results in obtaining an equal value for those sets that contain the same number of GMRs and aspects. In the running example, there is the same number of GMRs and aspects in rs_1 and rs_3, i.e., $|s_1| = 10$, $|s_3| = 10$, $\Psi(s_1) = 3$, and $\Psi(s_3) = 3$. Also, the number of common GMRs and aspects in these sets with user-selected ones is the same. Therefore, they obtained the same values for $Score_1$ and $Score_2$. However, these two sets are distinct. To reflect the difference of suggested sets, the distinctive characteristics of each GMR belonging to the sets should be considered. As an indicator to represent a difference between GMRs, we use the compatibility degree of each GMR (see Table 4). Thus, the average value of the compatibility degree of each GMR in the suggested compatible set is used as the third ranking criteria. For the given $rs_z = \{R_i, ..., R_m\}$, the average compatibility degree of Rs in rs_z is shown in Eq. 4.

$$Score_3(rs_z) = \Sigma_{j=i}^{m} f_d(R_j) \times \frac{1}{\Sigma(rs_z)} \tag{4}$$

In the example, $Score_3(rs_1) = 0.845$, $Score_3(rs_2) = 0.86$, and $Score_3(rs_3) = 0.89$.

Thus, the total rank for each rs_z is defined by Eq. 5.

$$Total\_Score(rs_z) = w_1 \times Score_1 + w_2 \times Score_2 + w_3 \times Score_3 \tag{5}$$

For our example, considering empirical values of 0.8, 0.1, and 0.1 for w_1, w_2, and w_3, respectively, the total score is achieved as $Total\_Score(rs_1) = 1.23$, $Total\_Score(rs_2) = 1.29$, and $Total\_Score(rs_3) = 1.24$. The values are normalized between 0 and 1 and presented in the descending order to the user. Fig. 2 shows the top-10 compatible sets, where the values for each set has been normalized and ordered in the GUI.

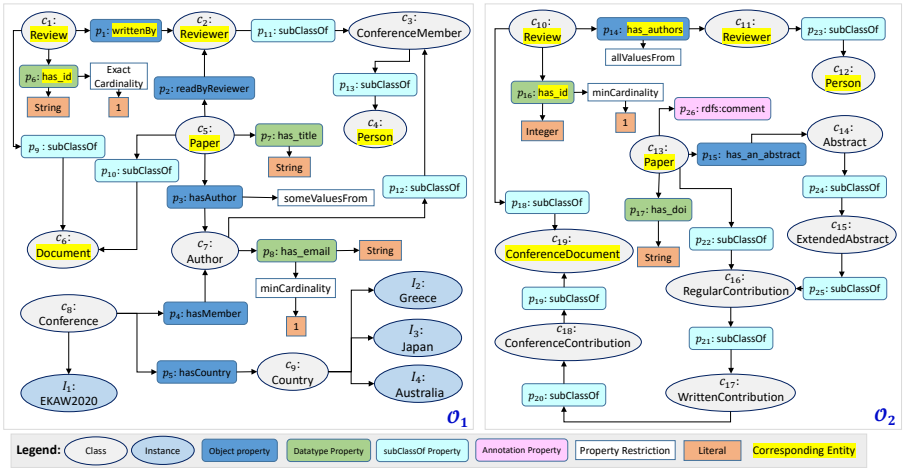

Fig. 3: Two sample source ontologies.

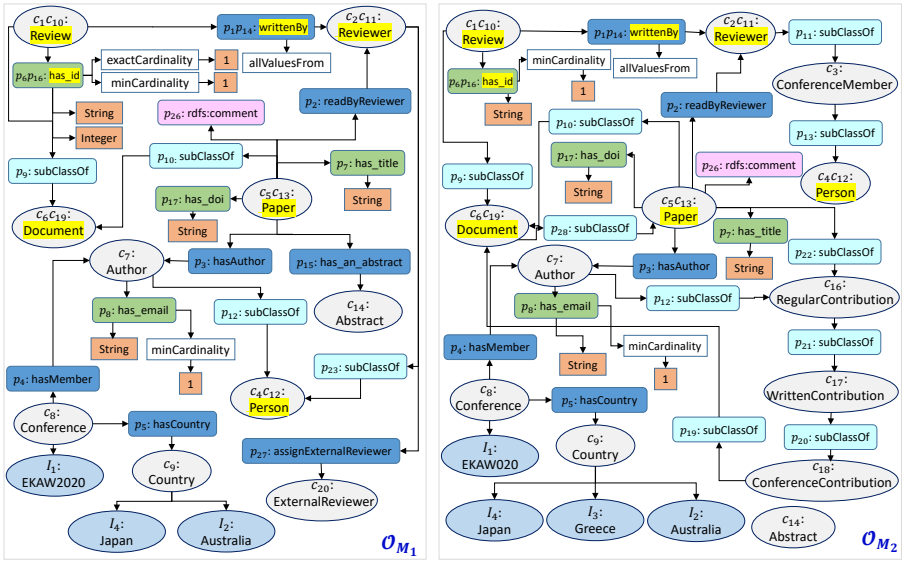

Fig. 4: Different versions of merged ontologies of Fig. 3.

5 Empirical Analysis

We have implemented the GMRs within the *CoMerger* [18][2]. In the ranking process, w_1, w_2, and w_3 have been empirically adjusted to 0.8, 0.1, and 0.1, respectively. For two sample source ontologies (see Fig. 3), we have created manually two different versions of the merged ontologies $\mathcal{O}_{M_1}$ and $\mathcal{O}_{M_2}$ (see

[2] Detail of GMR implementation: http://comerger.uni-jena.de/requirement.jsp

Fig. 4) to reflect different GMRs[3]. To this end, we analyze three user-selected GMRs and then discuss the extent to which they can be fulfilled simultaneously on $\mathcal{O}_{M_1}$ and $\mathcal{O}_{M_2}$.

First Use Case: $\mathcal{U} = \{R2, R3, R8, R16\}$. In $\mathcal{O}_{M_2}$, *R3* and *R8* are fulfilled. However, properties p_{15}, p_{24}, and p_{25} are missing, so *R2* is not fulfilled. Moreover, there is a cycle in $c_5c_{13} \sqsubseteq c_{16} \sqsubseteq c_{17} \sqsubseteq c_{18} \sqsubseteq c_6c_{19} \sqsubseteq c_5c_{13}$, which indicates that *R16* also is not fulfilled in $\mathcal{O}_{M_2}$. *R2* and *R16* are incompatible. Because *R2* will add the missing properties and will want to keep all properties. On the other side, *R16* will delete the is-a properties to be free of cycles. By applying *R2* in $\mathcal{O}_{M_2}$, properties p_{15}, p_{24}, and p_{25} will be added. Thus, all properties can be preserved at the merged ontology. However, by applying *R16*, property p_{28} will be deleted in order to be free of cycles. This action causes that *R2* failed. In this case, if *R2* wants to add p_{28}, a cycle will be generated. So, *R2* could not completely be fulfilled in the merged ontology. Three missing properties can be added, but one property (p_{28}) could not be preserved.

Thus, our system suggests as the best possible compatible set $rs_1 = \{$*R1*, *R3*, *R8*, *R9*, *R10*, *R11*, *R13*, *R14*, *R16*, *R17*, *R18*$\}$ and $rs_2 = \{$*R3*, *R8*, *R9*, *R10*, *R11*, *R13*, *R14*, *R15*, *R16*, *R17*, *R18*$\}$, in which the *R2* is not considered. These two sets have the same scores 1.0 based on our proposed method. Thus, given the user-selected GMRs, there is a superset of compatible GMRs that can be fulfilled simultaneously. The next possible compatible set is when *R16* is not considered and *R2* will be kept. Thus, the system suggests the set $rs_3 = \{$*R2*, *R3*, *R5*, *R7*, *R8*, *R10*, *R12*, *R13*, *R14*, *R19*, *R20*$\}$ with score 0.986. For the given $\mathcal{U}$, the 3^C-Cliques are $\{R3, R8, R16\}, \{R2, R8, R16\}$, and $\{R2, R3, R8\}$, and a 2^C-Clique is $\{R3, R8\}$. In Table 5, all $\mathcal{K}^C$-max-Cliques are shown, which are all possible maximal compatible sets for the user-selected GMRs. rs_1-rs_6, rs_8, and rs_9 are 11^C-max-Cliques, while rs_7, rs_{10}-rs_{16} are 10^C-max-Cliques, and rs_{17} and rs_{18} are 8^C-max-Clique and 7^C-max-Clique, respectively.

Second Use Case: $\mathcal{U} = \{R3, R6, R13\}$. In $\mathcal{O}_{M_1}$, *R3* is fulfilled. *R13* applies one type restriction. So, only one type for property p_6p_{16}`:has_id` should be preserved. But, applying *R13* will cause that *R6* will not be fulfilled, because not all values of property p_6p_{16} are preserved. Thus, *R6* and *R13* have a conflict with each other and cannot be fulfilled simultaneously in $\mathcal{O}_{M_1}$. Given the user-selected GMRs, there are two 2^C-Cliques as $\{R3, R13\}$ and $\{R3, R6\}$. Our method suggests as $\mathcal{K}^C$-max-clique $rs_1 = \{$*R2*, *R3*, *R5*, *R7*, *R8*, *R10*, *R12*, *R13*, *R14*, *R19*, *R20*$\}$ in which *R6* is not include. The next two maximum compatible sets are $rs_2 = \{R2, R3, R5, R7, R8, R9, R10, R12, R13, R14\}$, $rs_3 = \{R1, R2, R3, R5, R7, R8, R10, R13, R14, R19, R20\}$, respectively.

Third Use Case: $\mathcal{U} = \{R1, R2, R3, R8, R10, R19\}$. In $\mathcal{O}_{M_1}$, classes c_3, c_{13}-c_{15} are missing. By applying *R1*, these classes will be added to the $\mathcal{O}_{M_1}$. Moreover, properties p_{11}, p_{13}, p_{18}-p_{22}, p_{24}, and p_{25} are missing. Thus, applying *R2* will cause that these properties will be added to the $\mathcal{O}_{M_1}$. *R3* will add the missing instance I_2 to the $\mathcal{O}_{M_1}$. *R8*, *R10*, and *R19* are fulfilled in $\mathcal{O}_{M_1}$. In $\mathcal{O}_{M_2}$, class c_{15} and properties p_{15}, p_{24}, and p_{25} are missing. Applying *R1* and *R2* will

[3] Ontologies available at: https://github.com/fusion-jena/CoMerger/GMR

Table 5: All possible maximum compatible sets for user-selected GMRs $\mathcal{U} = \{R2, R3, R8, R16\}$. Green (no-line): user-selected compatible GMRs; Red (double-underline): user-selected incompatible GMRs; Orange(underline): extra compatible GMRs.

$\mathcal{RS}$	$\mathcal{K}$	Compatible	Incompatible	Score
rs_1	11	{*R1*, *R3*, *R8*, *R9*, *R10*, *R11*, *R13*, *R14*, *R16*, *R17*, *R18*}	{*R2*}	1.0
rs_2	11	{*R3*, *R8*, *R9*, *R10*, *R11*, *R13*, *R14*, *R15*, *R16*, *R17*, *R18*}	{*R2*}	1.0
rs_3	11	{*R2*, *R3*, *R5*, *R7*, *R8*, *R10*, *R12*, *R13*, *R14*, *R19*, *R20*}	{*R16*}	0.986
rs_4	11	{*R2*, *R3*, *R5*, *R7*, *R8*, *R9*, *R10*, *R11*, *R12*, *R13*, *R14*}	{*R16*}	0.975
rs_5	11	{*R1*, *R2*, *R3*, *R5*, *R7*, *R8*, *R10*, *R13*, *R14*, *R19*, *R20*}	{*R16*}	0.973
rs_6	11	{*R2*, *R3*, *R5*, *R7*, *R8*, *R10*, *R13*, *R14*, *R15*, *R19*, *R20*}	{*R16*}	0.973
rs_7	10	{*R3*, *R6*, *R8*, *R9*, *R10*, *R11*, *R15*, *R16*, *R17*, *R18*}	{*R2*}	0.97
rs_8	11	{*R1*, *R2*, *R3*, *R5*, *R7*, *R8*, *R9*, *R10*, *R11*, *R13*, *R14*}	{*R16*}	0.963
rs_9	11	{*R2*, *R3*, *R5*, *R7*, *R8*, *R9*, *R10*, *R11*, *R13*, *R14*, *R15*}	{*R16*}	0.963
rs_{10}	10	{*R1*, *R3*, *R6*, *R8*, *R9*, *R10*, *R11*, *R16*, *R17*, *R18*}	{*R2*}	0.957
rs_{11}	10	{*R2*, *R3*, *R5*, *R6*, *R7*, *R8*, *R10*, *R15*, *R19*, *R20*}	{*R16*}	0.944
rs_{12}	10	{*R2*, *R3*, *R5*, *R6*, *R7*, *R8*, *R10*, *R12*, *R19*, *R20*}	{*R16*}	0.943
rs_{13}	10	{*R2*, *R3*, *R5*, *R6*, *R7*, *R8*, *R9*, *R10*, *R11*, *R15*}	{*R16*}	0.934
rs_{14}	10	{*R2*, *R3*, *R5*, *R6*, *R7*, *R8*, *R9*, *R10*, *R11*, *R12*}	{*R16*}	0.933
rs_{15}	10	{*R1*, *R2*, *R3*, *R5*, *R6*, *R7*, *R8*, *R10*, *R19*, *R20*}	{*R16*}	0.931
rs_{16}	10	{*R1*, *R2*, *R3*, *R5*, *R6*, *R7*, *R8*, *R9*, *R10*, *R11*}	{*R16*}	0.921
rs_{17}	8	{*R3*, *R4*, *R8*, *R9*, *R10*, *R11*, *R13*, *R14*}	{*R2*, *R16*}	0.719
rs_{18}	7	{*R3*, *R4*, *R6*, *R8*, *R9*, *R10*, *R11*}	{*R2*, *R16*}	0.676

add the missing classes and properties in $\mathcal{O}_{M_2}$. *R3*, *R8*, and *R10* are fulfilled in $\mathcal{O}_{M_2}$. However, in the origin $\mathcal{O}_{M_2}$, the class c_{14} was unconnected. But, applying *R1* and *R2* caused that now c_{14} be connected. Thus, *R19* is fulfilled. In this case study, the user-selected GMRs are compatible with each other, however, a superset of other compatible GMRs with $\mathcal{U}$ is suggested. The maximum compatible sets are $rs_1 = \{R1, R2, R3, R5, R7, R8, R10, R13, R14, R19, R20\}$ and $rs_2 = \{R1, R2, R3, R5, R6, R7, R8, R10, R19, R20\}$.

6 Conclusion

Various ontology merging systems have been proposed. Each covers a group of Generic Merge Requirements (GMRs). Since not all GMRs can be fulfilled at the same time, we proposed a graph-based framework to systematically determine the GMRs compatibility interaction. The framework allows users to specify the most important GMRs for their specific task at hand and detects a maximal compatible superset. This result can then be used to select a proper merge method or to parameterize a generic merge method. The intuition behind using the graph theory is to facilitate the encoding of the GMRs' compatibility via the graph presentation and reveal the other possible compatible requirements. GMRs embedded in the proposed framework can be easily extended, in which building the GMR interaction's graph and obtaining their compatibility can be

performed in the same procedure for the new adapted GMRs. We have analyzed the GMRs within the $\mathcal{CoMerger}$ system, where the users can access to the logged information of applying GMRs on their merged ontologies. Through the proposed framework, potential conflicts between GMRs can be found. Not all of these potential conflicts will actually materialize in each concrete merged ontology. In future work, we will investigate how the approach can be extended to take this into account. Our second future plan is a user study about the extent to which the users agree with the ranked suggested sets.

Acknowledgments.

S. Babalou is supported by a scholarship from German Academic Exchange Service (DAAD).

References

1. R. A. Pottinger and P. A. Bernstein, "Merging models based on given correspondences," in *VLDB*, pp. 862–873, 2003.
2. S. Raunich and E. Rahm, "Target-driven merging of taxonomies with ATOM," *Inf. Syst.*, vol. 42, pp. 1–14, 2014.
3. E. Jiménez-Ruiz, B. C. Grau, I. Horrocks, and R. Berlanga, "Ontology integration using mappings: Towards getting the right logical consequences," in *ESWC*, pp. 173–187, Springer, 2009.
4. L. Chiticariu, P. G. Kolaitis, and L. Popa, "Interactive generation of integrated schemas," in *ACM SIGMOD*, pp. 833–846, 2008.
5. D. Thau, S. Bowers, and B. Ludäscher, "Merging taxonomies under rcc-5 algebraic articulations," in *ONISW*, pp. 47–54, ACM, 2008.
6. S. P. Ju, H. E. Esquivel, A. M. Rebollar, M. C. Su, *et al.*, "CreaDO–a methodology to create domain ontologies using parameter-based ontology merging techniques," in *MICAI*, pp. 23–28, IEEE, 2011.
7. M. Mahfoudh, L. Thiry, G. Forestier, and M. Hassenforder, "Algebraic graph transformations for merging ontologies," in *MEDI*, pp. 154–168, Springer, 2014.
8. N. F. Noy and M. A. Musen, "The prompt suite: interactive tools for ontology merging and mapping," *INT J HUM-COMPUT ST*, pp. 983–1024, 2003.
9. A. Makwana and A. Ganatra, "A known in advance, what ontologies to integrate? for effective ontology merging using k-means clustering," *IJIES, Vol.11, No.4*, 2018.
10. K. Saleem, Z. Bellahsene, and E. Hunt, "Porsche: Performance oriented schema mediation," *Information Systems*, vol. 33, no. 7, pp. 637–657, 2008.
11. A. Radwan, L. Popa, I. R. Stanoi, and A. Younis, "Top-k generation of integrated schemas based on directed and weighted correspondences," in *SIGMOD*, 2009.
12. N. M. El-Gohary and T. E. El-Diraby, "Merging architectural, engineering, and construction ontologies," *J COMPUT CIVIL ENG*, pp. 109–128, 2009.
13. G. Stumme and A. Maedche, "FCA-Merge: Bottom-up merging of ontologies," in *IJCAI*, pp. 225–230, 2001.
14. M. Priya and C. A. Kumar, "An approach to merge domain ontologies using granular computing," *Granular Computing*, pp. 1–26, 2019.

15. M. Priya and A. K. Cherukuri, "A novel method for merging academic social network ontologies using formal concept analysis and hybrid semantic similarity measure," *Library Hi Tech*, 2019.
16. A. Guzmán-Arenas and A.-D. Cuevas, "Knowledge accumulation through automatic merging of ontologies," *EXPERT SYST APPL*, pp. 1991–2005, 2010.
17. S. Babalou and B. König-Ries, "GMRs: Reconciliation of generic merge requirements in ontology integration," *In SEMANTICS Poster and Demo.*, 2019.
18. S. Babalou, E. Grygorova, and B. König-Ries, "CoMerger: A customizable online tool for building a consistent quality-assured merged ontology," in *In ESWC, Poster and Demo Track*, June, 2020.
19. L.-Y. Zhang, J.-D. Ren, and X.-W. Li, "OIM-SM: A method for ontology integration based on semantic mapping," *J INTELL FUZZY SYST*, 2017.
20. M. Fahad, N. Moalla, and A. Bouras, "Detection and resolution of semantic inconsistency and redundancy in an automatic ontology merging system," *JIIS*, vol. 39, no. 2, pp. 535–557, 2012.
21. M. Mahfoudh, G. Forestier, and M. Hassenforder, "A benchmark for ontologies merging assessment," in *KSEM*, pp. 555–566, 2016.
22. S. Raunich and E. Rahm, "Towards a benchmark for ontology merging.," in *OTM*, vol. 7567, pp. 124–133, 2012.
23. F. Duchateau and Z. Bellahsene, "Measuring the quality of an integrated schema.," in *ER*, pp. 261–273, 2010.
24. N. F. Noy, D. L. McGuinness, *et al.*, "Ontology development 101: A guide to creating your first ontology," 2001.
25. M. Poveda-Villalón, M. C. Suárez-Figueroa, and A. Gómez-Pérez, "Validating ontologies with oops!," pp. 267–281, 2012.
26. A. Rector, N. Drummond, M. Horridge, J. Rogers, H. Knublauch, R. Stevens, H. Wang, and C. Wroe, "Owl pizzas: Practical experience of teaching owl-dl: Common errors & common patterns," in *EKAW*, pp. 63–81, Springer, 2004.
27. E. Grygorova, S. Babalou, and B. König-Ries, "Toward owl restriction reconciliation in merging knowledge," in *In ESWC, Poster and Demo Track*, June, 2020.
28. E. Tomita, A. Tanaka, and H. Takahashi, "The worst-case time complexity for generating all maximal cliques and computational experiments," *TCS*, 2006.