Towards Multiple Ontology Merging with *CoMerger*

Samira Babalou, Birgitta König-Ries*

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

Abstract. To obtain a knowledge graph representing a domain of interest, it is often necessary to combine several, independently developed ontologies. Existing approaches are mostly limited to binary merge and lack scalability. This paper presents the development of an efficient multiple ontologies merging method, *CoMerger*. For efficient processing, rather than directly merging a large number of ontologies, we group related concepts across ontologies into partitions and merge first within and then across those partitions. Experiments on real-life datasets confirm the feasibility of our approach and demonstrate its superiority over binary strategies. Our implementation is available through a live web portal.

Keywords: Ontology merging . Partitioning . N-ary merge

1 Introduction

Ontologies represent the semantic model of data on the web. For many usecases, individual ontologies cover just part of or one specific perspective on the domain of interest. By merging them into one knowledge graph their complementarity can be leveraged. The merge process plays an important role in multiple different aspects of the Semantic Web (cf. [1]). Most existing approaches [2, 3] are limited to merging two ontologies at a time, due to using a *binary* merge. Merging $n > 2$ ontologies in a single step using a so called *n-ary* strategy [4], has not been extensively studied so far, but seems to be promising to obtain better scalability. We aim to investigate to which extent this is the case.

Let us consider five source ontologies with their correspondences depicted by dashed lines (Fig. 1). To estimate the merge effort, we measure combining the corresponding entities into an integrated entity $|combine|$, reconstructing the relationship $|reconst|$, and output generation $|output|$. In the binary-ladder strategy [4], a popular type of binary merge, first, O_1 and O_2 are combined into an O_{12}. Then, O_{12} is merged with O_3 and so on (see Fig. 1 and Table 1). The merged ontology obtained by an n-ary approach has the same structure as the final merged ontology of the binary-ladder merge for our example. The binary merge approach needs 10 combinations, 32 reconstructions, and 4 times output generation, while the n-ary method needs 6, 28, and 1, respectively. This means that for our example, the n-ary method reduces the effort to

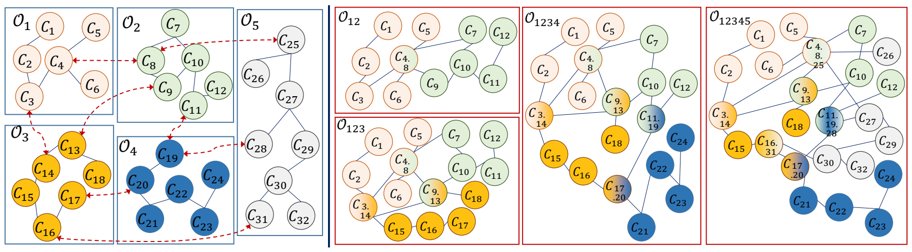

Fig. 1: Five ontologies with the merged ontologies in each step of binary merge.

Table 1: Number of operations for merging the five sample ontologies.

	N-ary	Binary				
	O_1 & O_2 & O_3 & O_4 & O_5	O_1 & O_2	O_{12} & O_3	O_{123} & O_4	O_{1234} & O_5	Total
$\lvert combine \rvert$	6	1	2	4	3	10
$\lvert reconst \rvert$	28	5	8	6	13	32
$\lvert output \rvert$	1	1	1	1	1	4

60, 87,5, and 25% of the effort needed with the binary method. The general pattern will be the same for other examples. The achieved improvements are significant compared to binary approaches, especially when dealing with a large number of ontologies.

To handle a large number of source ontologies, we aim to reduce the time and operational complexity while achieving at least the same quality of the final result compared to the binary merge or even improve upon it. In our n-ary method, $\mathcal{C}o\mathcal{M}erger$[1], we develop an efficient merging technique that scales to many ontologies. It takes as input a set of source ontologies $O_S = \{O_1, ..., O_n\}$ with a set of corresponding sets $\mathcal{CS}$ extracted from their mappings and generates a merged ontology O_M. At first, the n source ontologies are partitioned into k blocks ($k << n$). After that, the blocks are individually refined and finally combined to produce the merged ontology followed by a global refinement.

2 Proposed Method

Our method implemented in $\mathcal{C}o\mathcal{M}erger$ consists of three main phases: initialization, partitioning, and combining. Mappings between source ontologies can be given or can be created with a matching tool [5] embedded in $\mathcal{C}o\mathcal{M}erger$ in a preprocessing step. These mappings consists of sets of corresponding entities. The relationship determining correspondence can be equality, similarity, or subset (is-a) types. In $\mathcal{C}o\mathcal{M}erger$, we consider the similarity type with at least a given similarity value (1:1 mapping). We maintain a model $\mathcal{M}$ which keeps the information across a group of correspondences over multiple ontologies. We denote the cardinality of each corresponding set by $Card(cs_j)$, where $cs_j \in \mathcal{CS}$.

(1) Initialization Phase. We build an initial merge model $\mathcal{I}_M$ and parse the source ontologies by extracting all entities into the $\mathcal{I}_M$. After that, the list of corresponding

[1] The tool as well as all data used for evaluation are available online at `http://comerger.uni-jena.de` and `https://github.com/fusion-jena/CoMerger`

sets $\mathcal{CS}$ from the given mappings are processed to build the model of mappings $\mathcal{M}$ over multiple ontologies. For each entry of $\mathcal{M}$, a new integrated entity in $\mathcal{I}_M$ is created. To construct the initial relations between the entities in $\mathcal{I}_M$, all axioms from the source ontologies should be processed. Thus, we translate the source ontologies' axioms concerning the corresponding sets that exist in $\mathcal{M}$. If an axiom's entities have a corresponding entity in $\mathcal{M}$, the axiom is translated with the generated integrated entity; if not, the axiom is imported into $\mathcal{I}_M$ without any changes.

(2) Partitioning Phase. An ontology block $\mathcal{L}$ is a subset (or whole) of one (or more) source ontologies with no overlap with other blocks. The number of blocks is denoted by k and $C\mathcal{L} = \{\mathcal{L}_1, ..., \mathcal{L}_k\}$ is the set of all blocks. We decompose the ontologies merging task $Merge(O_1, ..., O_n)$ to the blocks merging task $Merge(\mathcal{L}_1, ..., \mathcal{L}_k)$, $k << n$. To achieve this, we use a set of *pivot* $\mathcal{P}$ classes:

(i) Finding pivot classes $\mathcal{P}$. To find the pivot set of classes $\mathcal{P}$ for the divide process, we measure a *reputation* degree of each class belonging to $\mathcal{CS}$, as $reputation(c_t) = Card(c_t) \times Conn(c_t)$. It is based on the number of corresponding classes $Card(c_t)$ along with their connectivity degree $Conn(c_t)$ from the respective source ontologies. The classes with high $Card(c_t)$ values in $\mathcal{CS}$ have the best overlap within O_S. However, contemplating only this metric tends to choose isolated classes as the pivot classes. Thus, class connectivity has been taken into account. The connectivity degree $Conn(c_t)$ of a class is indicated by the number of associated taxonomic and non-taxonomic relations for that class. Thus, $\mathcal{P}$ is realized by a sorted list of $\mathcal{CS}$'s elements based on their *reputation* degrees.

(ii) Partitioner. This step divides all classes from $\mathcal{I}_M$ into a set of blocks $C\mathcal{L} = \{\mathcal{L}_1, ..., \mathcal{L}_k\}$ based on a structural-based similarity, in which classes close in the hierarchy are considered similar and should be placed in the same block. Thus, once a class is assigned to a block, all its adjacent classes consequently will also be added. The first block $\mathcal{L}_1$ is created by picking the first $\mathcal{P}$'s element, which has the highest reputation degree. For each corresponding class of $\mathcal{P}$'s element, all classes with their adjacent classes on their respective ontologies are added to the block. Then, the next element of $\mathcal{P}$ is selected to create a new block, if at least one of its classes has not been assigned to the previous blocks. This process is continued until all elements of $\mathcal{P}$ are processed. The overall number of blocks is determined based on the number of $\mathcal{P}$'s elements and the amount of shared classes between $\mathcal{P}$'s elements.

(3) Combining Phase. This phase is split into two steps. In the *intra-combination* step, the entities inside the blocks are combined to create local sub-ontologies. We retrieve all existing axioms from $\mathcal{I}_M$. Each class is augmented by the original or translated properties axioms, including taxonomic and non-taxonomic relations. We assign each axiom to a block in which at least all its entities are contained. If the classes of each axiom are distributed across multiple blocks, they are not added to any block and are marked as distributed axioms $dist_{axiom}$. Finally, the local refinement process takes place for each sub-ontology. To further improve performance, this step can utilize parallel processing. In the *inter-combination* step, the final merged ontology O_M is constructed. We follow a sequential merging based on the inter-block relatedness, $|dist_{axiom}(\mathcal{L}_i) \cap dist_{axiom}(\mathcal{L}_j)|$. It is based on the number of shared distributed axioms between two blocks $\mathcal{L}_i$ and $\mathcal{L}_j$. At first, the two blocks with the highest inter-block

Table 2: Comparing n-ary (N), balanced (B), and ladder (L) merge strategies.

	dataset$_4$			*dataset*$_6$			*dataset*$_{10}$			*dataset*$_{11}$			*dataset*$_{12}$		
	N	B	L	N	B	L	N	B	L	N	B	L	N	B	L
time	1.3	3.5	4.5	1.8	7	8.3	61.8	252.5	507	62.3	230.3	592.2	150.7	806.3	4738.8
$\|Cor\|$	47	65	64	82	127	128	266	368	351	267	387	387	1139	2605	2154
$\|tr\|$	790	1270	1339	1310	2784	3462	6949	15773	31500	6960	16019	35458	30035	71884	153879
$\|R_G\|$	21	27	23	41	48	51	143	167	191	143	179	200	977	1429	1553
$\|Mer.\|$	1	3		1	6		1	16		1	18		1	54	

similarity value are merged. This includes adding all distributed axioms to them. Then, the next block with the highest inter-block relatedness will be selected to be merged. This process is continued until all blocks are merged. A set of global refinement will be applied on the last combined block. Upon that, the merged ontology O_M is built. To apply local and global refinement within $\mathcal{C}o\mathcal{M}erger$, we include a list of General Merge Requirements (GMR)s [6], containing, e.g., entities preservation, one type restriction, acyclicity, and connectivity.

3 Experimental Evaluation

We have selected sets of ontologies frequently used for evaluation tasks (see our repository), with $134 \leq |axioms| \leq 30364$ and $2 \leq n \leq 55$ and have created mappings with SeeCOnt [5]. To demonstrate the feasibility of our n-ary strategy, we compared it with binary ladder and balanced strategies [4]. The runtime performance (in seconds) for each merge strategy is shown in the third row (*time*) of Table 2. The n-ary merge is on average 4 (9) times faster than the balance (ladder) binary merge, respectively. Moreover, we present the operation complexity of these three strategies in Table 2. In each test of a binary merge, only the correspondence between two entities can be integrated simultaneously into a new entity. However, in n-ary merge, the corresponding entities from multiple source ontologies can be integrated into the new entities. Thus, the number of total corresponding entities $|Cor|$ in the binary merge is much higher than in the n-ary approach. Consequently, the required amount of combining them into new entities and translating their axioms $|tr|$ is high. For global refinement, in most cases, the n-ary approach requires less actions compared to binary merges. We also compare the number of merge processes $|Mer.|$. While the n-ary approach only uses one iteration, ladder and balance methods require $n-1$ merge processes. Overall the result demonstrates that the n-ary approach has better (runtime and complexity) performance than binary approaches. In terms of quality, the n-ary approach achieves comparable results to its binary counterparts (see full result in our repository).

4 State of the Art

Merging strategies have been divided into two main categories [4]: *"binary"* and *"n-ary"*. The *binary* approach allows the merging of two ontologies at a time, while the *n-ary* strategy merges n ontologies ($n > 2$) in a single step. Most methodologies in the literature, such as [2, 3] agree on adopting a *binary* strategy. The few existing n-ary approaches [7, 8] suffer certain drawbacks: In [8], the final merge result depends on the order in which the source tree-structured XML schemas are matched and merged. In [7], the experimental tests were carried out on a few small source ontologies, only. Despite the efforts of these research studies, developing an efficient, scalable n-ary method has not been practically applied.

5 Conclusion

Existing ontology merge approaches scale rather poorly to the merging of multiple ontologies. We proposed the n-ary multiple ontologies merging approach *CoMerger* based on a partitioning based method, to overcome this issue. Taking advantage of the parallelization is on our future research agenda.

Acknowledgments

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

References

1. M. T. Finke, R. W. Filice, and C. E. Kahn Jr, "Integrating ontologies of human diseases, phenotypes, and radiological diagnosis," *J AM MED INFORM ASSN*, vol. 26, no. 2, pp. 149–154, 2019.
2. M. Priya and C. A. Kumar, "An approach to merge domain ontologies using granular computing," *Granular Computing*, pp. 1–26, 2019.
3. S. Raunich and E. Rahm, "Target-driven merging of taxonomies with ATOM," *Inf. Syst.*, vol. 42, pp. 1–14, 2014.
4. C. Batini, M. Lenzerini, and S. B. Navathe, "A comparative analysis of methodologies for database schema integration," *In CSUR*, 1986.
5. A. Algergawy, S. Babalou, M. J. Kargar, and S. H. Davarpanah, "SeeCOnt: A new seeding-based clustering approach for ontology matching," in *ADBIS*, 2015.
6. S. Babalou and B. König-Ries, "GMRs: Reconciliation of generic merge requirements in ontology integration," *In SEMANTICS Poster and Demo.*, 2019.
7. N. Maiz, M. Fahad, O. Boussaid, and F. Bentayeb, "Automatic ontology merging by hierarchical clustering and inference mechanisms," in *I-KNOW*, pp. 1–3, 2010.
8. K. Saleem, Z. Bellahsene, and E. Hunt, "Porsche: Performance oriented schema mediation," *Inf. Syst.*, vol. 33, no. 7, pp. 637–657, 2008.