Automated Clustering of Metamodel Repositories

Francesco Basciani[1], Juri Di Rocco[1], Davide Di Ruscio[1](✉), Ludovico Iovino[2], and Alfonso Pierantonio[1]

[1] Department of Information Engineering, Computer Science and Mathematics, Università degli Studi dell'Aquila, L'Aquila, Italy
francesco.basciani@graduate.univaq.it, {juri.dirocco,davide.diruscio, alfonso.pierantonio}@univaq.it

[2] Gran Sasso Science Institute, L'Aquila, Italy
ludovico.iovino@gssi.infn.it

Abstract. Over the last years, several model repositories have been proposed in response to the need of the MDE community for advanced systems supporting the reuse of modeling artifacts. Modelers can interact with MDE repositories with different intents ranging from merely repository browsing, to searching specific artifacts satisfying precise requirements. The organization and browsing facilities provided by current repositories is limited since they do not produce structured overviews of the contained artifacts, and the ategorization mechanisms (if any) are based on manual activities. When dealing with large numbers of modeling artifacts, such limitations increase the effort for managing and reusing artifacts stored in model repositories. By focusing on metamodel repositories, in this paper we propose the application of clustering techniques to automatically organize stored metamodels and to provide users with overviews of the application domains covered by the available metamodels. The approach has been implemented in the MDEForge repository.

Keywords: Model Driven Engineering · Model repositories · Metamodel clustering · MDEForge

1 Introduction

The increasing adoption of Model-Driven Engineering (MDE) [1] in business organizations led to the need for gathering artifacts in model repositories [2]. Several model repositories (see [3–6] just to mention a few) have been introduced in the past decade. Among them *metamodel zoos* (as for instance the Ecore Zoo[1]) hold metamodels, which are typically categorized to improve search and/or browse operations. However, locating relevant information in a vast repository is intrinsically difficult, because it requires domain experts to manually annotate *all* metamodels in the repository with accurate metadata [7]: an activity that is

[1] ATLAS Ecore Zoo: http://www.emn.fr/z-info/atlanmod/index.php/Zoos.

© Springer International Publishing Switzerland 2016

time consuming and prone to errors and omissions. In fact, acquiring knowledge about a software artifact is a challenging task: it is estimated that up to 60 % of software maintenance is spent on comprehension [8].

Software clustering [9] is a well-established discipline, which has found numerous applications in reverse engineering and software maintenance. It promotes the automated categorization of software artifacts (like functions, classes, or files) into high-level structures based on their similarity distance [10]. Software clustering is applied, for instance, to detect misplaced software artifacts [11].

In this paper, in order to mitigate the difficulties related to the manual categorization of metamodels, we propose the application of clustering techniques for metamodel repositories able to automatically organize metamodels into clusters. Mutually similar metamodels are grouped together depending on a proximity measure, whose definition can be given according to specific search and browsing requirements. The approach is based on agglomerative hierarchical clustering [12] (see Sect. 3) and explores well-known proximity measures as well as metamodel-specific ones, each providing different browsing characteristics. The method has been already implemented in the MDEForge repository [13] and it is available online[2]. Furthermore, an evaluation has been conducted by considering different similarity measures, each characterized by specific accuracy and performance indexes. The evaluation permitted also to identify the application domains represented by the metamodels stored in an arbitrary repository.

This paper is structured as follows. In Sect. 2 we illustrate the main characteristics of modeling repositories. Section 3 provides an overview about clustering techniques. In Sect. 4, we present our approach and show how it is able to automatically categorize metamodel repositories. In the next section, the approach is evaluated by applying it to a corpus of about 300 metamodels. A discussion about the results is provided in Sect. 6. In Sect. 7 related work is discussed and finally, in Sect. 8, we conclude and outline future plans.

2 Repositories of Modeling Artifacts in MDE

Modelers can interact with model repositories for several purposes. For instance, as in the case of source code repositories, users can be interested in acquiring knowledge from already developed modeling artifacts that might represent precious know-how to be conveyed to new modelers. Users that have clear requirements about the desired modeling artifacts, can use model repositories with the aim of finding the modeling artifacts that best fit the user needs. Whatever the modeler intents, repositories should provide users with a dedicated support for properly organizing the contained artifacts, and to effectively search and retrieve them. Especially in the case of large repositories, the potential benefits related to the availability of reusable artifacts might be missed if they cannot be suitably discovered. By considering the ways artifacts are organized, and thus the provided searching and browsing functionalities, it is possible to identify different kinds of repositories as discussed below.

[2] MDEForge site: http://www.mdeforge.org.

Flat Repositories: They are not structured in the sense that contained artifacts are not categorized, and searching and browsing functionalities are not available. With this kind of repositories we refer to any publicly available collection of artifacts that are stored in a file-system like manner. For instance, it's plenty of GitHub repositories containing modeling artifacts. Modelers are obliged to download and inspect such models in order to gain some insights about them and to check if they might satisfy their requirements.

Manually Classified Repositories Without *Searching and Browsing Functionalities:* Artifacts are manually categorized, and searching and browsing functionalities are not available. For instance, the Ecore Zoo repository consists of an organized list of metamodels. For each metamodel, different attributes are given including a short description about it, and a list of domains where the metamodel can be applied. To search for a specific metamodel in the repository, modelers have to rely on the in-page search facility of the used Web browser. Thus, when searching for specific words, modelers have to manually check where they occur (e.g., in the description, or in the domain attributes). For instance, in the case of the Ecore Zoo, different metamodels are available for supporting the management of projects in organizations, e.g., how to assign tasks and resources. Such metamodels can be identified by searching the string *"project management"* in the Ecore Zoo web page. In addition, also the Maven[3] metamodel is found, as the searched string occurs in its description, although Maven is a building tool thus it refers to a different domain.

Manually Classified Repositories with *Searching and Browsing Functionalities:* Artifacts are manually categorized, searching and browsing functionalities are available. Similarly to repositories like Ecore Zoo, artifacts are manually classified according to a predefined set of labels. The available searching facility permits to give search strings as input and match them against the description field of the available artifacts (e.g., see [2]).

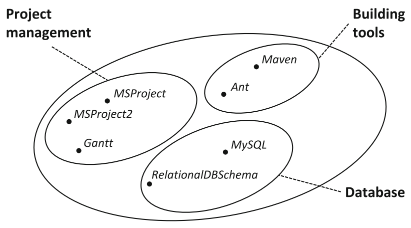


Fig. 1. Example of classified metamodels

Most of the potential benefits of such repositories remain unexploited especially when hundreds or even thousands of modeling artifacts have to be managed. In particular, by focusing on the provided functionalities for organizing, browsing, and searching *metamodels*, all the available repositories are affected by the following challenges:

C1. they do not provide the means to automatically produce structured overviews of the contained metamodels, which are typically shown as merely lists of stored elements, and which are consequently difficult to browse. Organizations like the one shown in Fig. 1 would permit to have an overview of the metamodels stored in the considered repository, e.g., with respect to the covered application domains;

[3] http://maven.apache.org/.

C2. all the available repositories do not provide mechanisms to automatically categorize the stored artifacts, by making the interaction with the repositories complex. Even users that want to contribute with additional artifacts have to manually annotate and classify them during the creation phase. This activity is subject to errors and inaccuracies especially for large repositories, and can compromise the searchability of artifacts. By considering the example shown in Fig. 1, it would be extremely relevant having a mechanism able to automatically assign the metamodel being added to one of the currently available categories (e.g., *building tools*, *database*, and *project management*) or even create a new one if none of them are appropriate.

In the next sections we propose techniques and tools to address these challenges by focusing on the management of metamodels stored in publicly available repositories. We propose the application of an unsupervised metamodel clustering mechanism, which permits to automatically organize unstructured metamodel repositories, and provides the users with overviews of the available metamodels. Thus *manually annotating metamodels is no longer necessary*, since the provided approach is able to automatically catogorize metamodels according to their content and structure.

3 Overview of Clustering Techniques

Clustering is one of the techniques for doing data mining and can be defined as the *process of organizing objects into groups of similar objects* [14]. A cluster is therefore a collection of objects, which are similar between them and are dissimilar to the objects belonging to other clusters [12]. Clustering is also known as *unsupervised classification* since we do not know a priori neither the number of classes nor their attributes. Clustering techniques are applied in a wide spectrum of areas including *biology* to classify plants and animals according to their properties, and *geology* to classify observed earthquake epicenters and thus to identify dangerous zones. Clustering has found numerous applications to *software* as well [9], where it is used in reverse engineering and software maintenance for categorizing software artifacts in many respects. Over the years several clustering methods have been developed like the hierarchical and partitional ones [12]. In the remainder of the section we focus on the hierarchical clustering technique since it underpins the approach proposed in the next section.

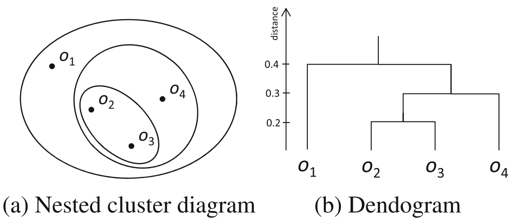


Fig. 2. Explanatory hierarchical clustering example

Hierarchical clustering produces a nested set of groups based on a criterion for merging or splitting clusters based on similarity. The nested grouping and similarity levels obtained by means of hierarchical algorithms are typically represented by means of *dendrograms* like the one in Fig. 2b.

By cutting the dendrogram at a desired distance *threshold*, a clustering of the data objects into disjoint groups is obtained. For instance, by considering the example in Fig. 2 if we choose as distance threshold the value 0.25 we obtain three clusters, i.e., $\{o_1, \{o_2, o_3\}, \{o_4\}\}$. With the threshold value 0.35 the obtained clusters are $\{o_1, \{o_2, o_3, o_4\}\}$. Hierarchical clustering methods can be *agglomerative* or *divisive*. The former starts with one object clusters and recursively merges two or more of the most similar clusters. The latter starts with a single cluster consisting of all the elements in the source data set and recursively split the clusters according to some criterion for obtaining at the end of the process a partition of one object clusters (named singleton clusters hereafter) [14].

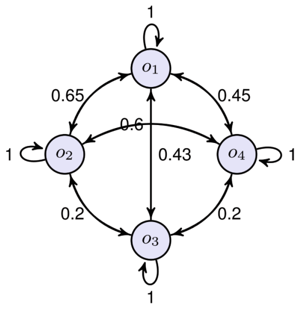
Algorithm 1. Basic agglomerative hierarchical clustering algorithm

```
1: Compute the proximity graph;
2: Merge the closest (most similar) two clusters;
3: Update the proximity matrix to reflect the proximity between the new cluster and the original
       clusters;
4: Repeat steps 3 and 4 until only a single cluster remains;
```

All the existing clustering methods share the fact that they can be applied when it is possible to specify a *proximity (or distance) measure* that permits to assess if elements to be clustered are mutually similar or dissimilar. The basic idea is that the *similarity level of two elements is inversely proportional to their distance*. The definition of the proximity measure plays a key role in any clustering method and it depends on many factors including the considered application domain, available data, and goals. Once the proximity measure is defined, it is possible to produce a proximity matrix, which is an n by n matrix (where n is the number of objects to be clustered) containing all the pairwise similarities or dissimilarities between the considered objects. For instance, by considering a simple data set O consisting of the objects o_1, o_2, o_3, and o_4, a corresponding proximity matrix based on a sample similarity function $s : O \rightarrow [0, 1]$ can be given like the one shown in Fig. 3a.

	o_1	o_2	o_3	o_4
o_1	1	0.65	0.43	0.45
o_2	0.65	1	0.2	0.6
o_3	0.43	0.2	1	0.3
o_4	0.45	0.6	0.3	1

(a) Proximity matrix



(b) Proximity graph

Fig. 3. Sample proximity matrix and graph

A proximity matrix induces the definition of a weighted graph (like the one shown in Fig. 3b) where nodes are the objects being clustered, and weighted edges represent the similarities between the connected objects. The availability of proximity graphs permits to see the clustering problem from a graph point of view as that of breaking the graph into connected components, one for each cluster [15]. By focusing on the *agglomerative hierarchical clustering*, a basic algorithm would use the proximity graph as shown in Algorithm 1 borrowed from [16]:

It is worth noting that to execute such a clustering algorithm a notion of *cluster proximity* is necessary to execute the second step of the algorithm. To this end, three different definitions of cluster distance can be used, namely *single link*, *complete link*, and *group average* [16]. With the single (complete) link definition, the proximity of two clusters is defined as the minimum (maximum) distance between any two objects in the considered clusters. Whereas, with the group average technique, the proximity of two clusters is the average pairwise proximities of all pairs of objects from the considered clusters.

In the next section we discuss how the traditional clustering concepts previously outlined can be employed in the context of metamodel repositories.

4 Proposed Metamodel Clustering Approach

In order to deal with the issues discussed in Sect. 2 in this section we show how to apply clustering techniques in the domain of model-driven engineering. The proposed approach is able to automatically organize metamodels stored in a given repository by analysing their content and not their metadata that might be erroneous or misplaced. An overview of the approach is given in Sect. 4.1, its implementation is presented in Sect. 4.2.

4.1 Overview

The main functionalities of the proposed clustering approach are shown in Fig. 4. It is important to remark that the figure shows only the functionalities strictly related to the automated classification of metamodels. For an overview about the typically provided functionalities of existing model repositories, interested readers can refer to [17]. As shown in Fig. 4, two different user roles are involved in the proposed clustering approach namely the *Repository Maintainer* and the *Repository User* discussed in the following.

Repository Maintainer: The application of the whole metamodel clustering approach is performed by the maintainer of the repository who can have access to the functionalities described below.

Apply Metamodel Clustering: It represents the key functionality of the proposed clustering approach. It consists of calculating the proximity matrix (as shown in Sect. 3) representing the similarities of all the metamodels available in the repository, and then applying the clustering algorithm in Algorithm 1.

Manage Singleton Clusters: When a new metamodel is being added to the repository, it may happen that according to the used proximity measure it does not fit in any of the existing clusters and consequently it induces the creation

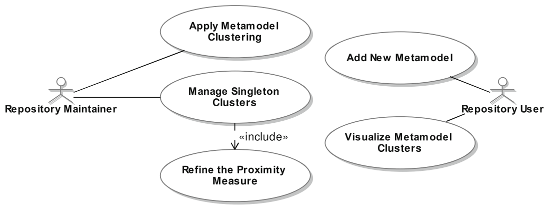

Apply Metamodel Clustering
Add New Metamodel
Repository Maintainer
Manage Singleton Clusters
Repository User
«include»
Visualize Metamodel Clusters
Refine the Proximity Measure

Fig. 4. Actors and use cases of the proposed approach

of a singleton cluster, i.e., a cluster consisting of only one element. The repository maintainer can periodically consider the available singleton clusters and verify if they have been created, e.g., because the used proximity measure has to be refined.

Refine the Proximity Measure: The proximity measure plays a key role in the whole clustering approach, and consequently its definition is an iterative process, aiming at increasing the accuracy of the automatically obtained metamodel clusters. The refinement process relies on the availability of reference data, which are typically obtained by manual activities. Such data must be approximated by the automated clustering procedure as discussed in the next section.

Repository User: Similarly to what happens in the case of open source software, the availability of public model repositories can give place to multitudes of users and developers that are willing to share their modeling artifacts. In this respect, by focusing on the metamodel clustering aspects, the proposed approach provides the users with the functionalities discussed below.

Add New Metamodel: In contrast with existing metamodel archives, users that add new metamodels in the repository can omit the specification of corresponding metadata. Even in such cases, the provided approach is able to automatically classify the new metamodels. In fact the appropriate clusters are identified by considering the content of the metamodels without the need for additional user input. However, as previously mentioned, it might happen that newly added metamodels do not fit in any of the existing clusters. Then, the repository maintainer takes care of such situations by means of the functionality *Manage Singleton Clusters* shown in Fig. 4.

Visualize Metamodel Clusters: The approach produces overviews of the automatically produced metamodel clusters. Thus in addition to the list of available metamodels, the system is able to generate graphical representations of the available metamodel clusters, and gives also the means to navigate them and to retrieve detailed information about their content if requested by the user.

4.2 Supporting Tool

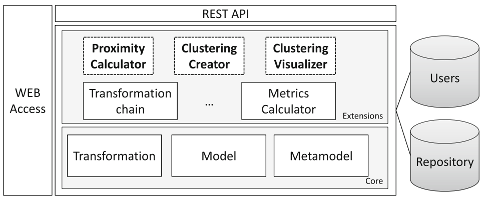


Fig. 5. MDEForge architecture

The proposed clustering method has been implemented as extensions in the MDEForge platform. In particular, as shown in Fig. 5, MDEForge consists of *core* services that are provided to enable the management of modeling artifacts, namely transformations, models, and metamodels. Atop of such core services, *extensions* can be developed to add new functionalities. For instance, in [18] we propose a service to automatically compose model transformations according to user requirements. We have also developed extensions to calculate several metrics on stored artifacts, and to support the understanding of metamodel and transformation characteristics [19,20].

In the remainder of the section, we give details about the extensions that are shown in Fig. 5 in bold and that we have developed to support the proposed clustering approach. Concerning the other services of MDEForge the reader can refer to [13,17].

Proximity Calculator: It plays a key role in the proposed clustering approach since it is responsible of calculating the mutual similarities between all the metamodels and thus create a corresponding proximity matrix like the one shown in Fig. 3a. Such calculations rely on the definition of a given similarity measure. As discussed in Sect. 3 identifying the appropriate similarity measure is a difficult task that might depend on the available data set, on the considered application domain, on the goal of the analysis being performed, etc. [12]. Consequently, from an architectural point of view, the proximity calculator has been designed in terms of an interface consisting of a method `calculateSimilarity(Metamodel` mm_1`, Metamodel` mm_1`)`, and then different concrete implementations can be provided. So far we have developed different similarity measures already available in the system even though we plan to experiment and provide additional ones. In particular, several similarity measures have been proposed in literature [14]. Among those typically applied to text documents we have considered the *cosine similarity* [14] and the *Dice's coefficient* [21] with the aim of relating the similarity of two metamodels on the terms used therein and consequently on the corresponding application domains. In particular:

– *Cosine similarity:* given two documents represented as term vectors, the similarity of the input documents corresponds to the correlation between the derived vectors. Such a correlation is calculated as the cosine of the angle between vectors [14]. In order to apply such a similarity measure on metamodels, for each of them we derive the corresponding string by borrowing the serialization mechanisms available in EMF-REST[4];
– *Dice's coefficient:* it is defined as twice the number of common bigrams (i.e., pairs of adjacent letters in the string) in the compared strings divided by the total number of terms in both strings [21]. Similarly to the application of the cosine similarity, to calculate the Dice's coefficient between two metamodels, we first derive a string serialization of them.

We have developed also two additional similarity functions specifically conceived for modeling artifacts. Both of them rely on the matching model calculated by means of EMFCompare[5]:

– *Match-based similarity:* it is defined as the total number of matched elements identified by EMFCompare divided by the total number of elements contained in the analysed couple of metamodels;
– *Containment-based similarity:* the previous index does not perform well when one of the input metamodels is contained in the other one. As an example we

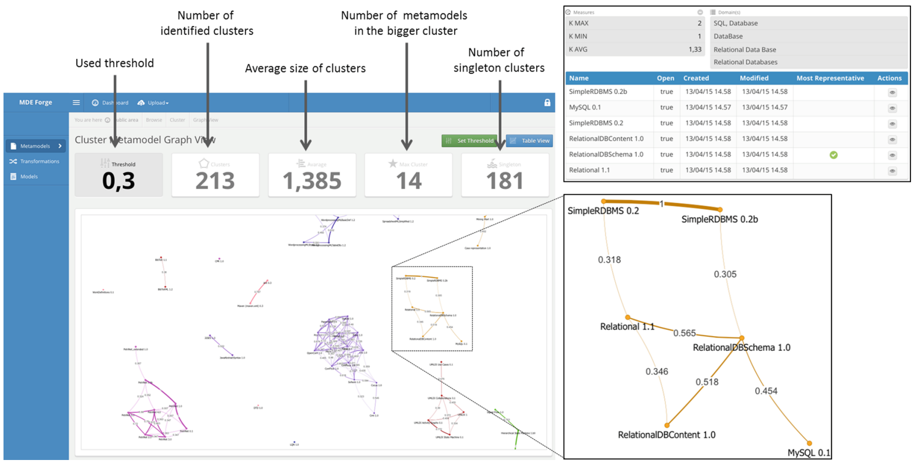
[4] http://emf-rest.com/.
[5] http://www.eclipse.org/emf/compare/.

can consider the full specification of UML and the UML Class Diagrams. In such cases the match-based similarity value would be very low since the total number of matched elements would be much lesser than the total number of elements contained in the two metamodels. In order to deal with such cases, the containment-based similarity is defined as the total number of matched elements divided by the lesser of the total elements in the two input metamodels.

The application on a concrete data set of the measures used by the proximity calculator is in-depth discussed in the next section.

Clustering Creator: By using the proximity calculator previously discussed, it creates clusters of metamodels by applying the agglomerative hierarchical clustering algorithm shown in Algorithm 1. As to the cluster proximity calculation, which is performed during each iteration of the algorithm, it is possible to specify the distance to be used, i.e., single link, complete link, and group average (see Sect. 3).

Cluster Visualizer: It creates graphical and tabular representations of the calculated metamodel clusters. The user can explore the available metamodels by specifying the similarity measure to be applied, and the threshold value used to filter the identified metamodels pairs and show only those that have a similarity value greater than the given threshold. The left hand side of Fig. 6 shows the cluster visualizer at work. In particular, the shown connected graphs represent the identified clusters and the thickness of the edges is proportional to the proximity value of the connected metamodels represented as nodes in the graph. For each cluster, the system permits to retrieve additional information as shown in the upper right-hand side of Fig. 6. In particular, given a cluster all the contained metamodels are listed together with additional information like



Measures	
K MAX	2
K MIN	1
K AVG	1,33

Domain(s)
SQL, Database
DataBase
Relational Data Base
Relational Databases

Name	Open	Created	Modified	Most Representative	Actions
SimpleRDBMS 0.2b	true	13/04/15 14.58	13/04/15 14.58		
MySQL 0.1	true	13/04/15 14.57	13/04/15 14.57		
SimpleRDBMS 0.2	true	13/04/15 14.58	13/04/15 14.58		
RelationalDBContent 1.0	true	13/04/15 14.58	13/04/15 14.58		
RelationalDBSchema 1.0	true	13/04/15 14.58	13/04/15 14.58	✓	
Relational 1.1	true	13/04/15 14.58	13/04/15 14.58		

Fig. 6. Sample visualizations of automatically created metamodel clusters

the most representative metamodel, i.e., the one most connected with the other ones in the cluster. Additionally, metamodels can be downloaded or even viewed by means of an integrated tree-based editor.

It is important to remark that the platform is modular in the sense that interested developers can extend or even substitute the provided services, for instance, to refine the proximity measures if the proposed ones do not properly meet their requirements.

5 Evaluation of the Proposed Metamodel Clustering Approach

In this section we discuss the application of the clustering approach on a concrete data set consisting of 295 metamodels retrieved from the Ecore Zoo. The main goal of this section is to discuss the ability of the clustering method *(i)* to automate the creation of metamodel groups according to different similarity measures, and *(ii)* to provide the user with organized and interactive views of the metamodel repository. We have applied the clustering technique by using the four similarity functions discussed in the previous section and by specifying different thresholds.

In order to evaluate the calculated clusters we have applied a clustering validation technique based on *external criteria* [22]: the final goal is to validate the results of the employed clustering technique comparing them with the manually pre-specified clusters.

Experimental Setting and Dataset. We have downloaded all the metamodels from the Ecore Zoo and we have manually grouped them with respect to their content and the domain descriptions when available. Thus, we have individually analysed the metamodels and incrementally defined the metamodel clusters. At the end of this manual process, that took about two working days of one senior researcher with a consolidated expertise in metamodeling and model-driven engineering, we identified 90 groups, including 43 singleton[6]. Each group represents a specific application domain, e.g., database management, project management, building tools, and model transformation languages. Subsequently, we have applied on the downloaded metamodels the proposed clustering approach by using the four similarity functions previously discussed and by shifting the thresholds from the value of 0.1 to 1.0 with 0.05 steps. Clearly, a too low threshold corresponds to consider the repository population almost undistinguished, whilst a too high threshold returns too many clusters with too few elements. The manually identified clusters and those automatically created are analysed as discussed later on this section.

[6] All the manipulated data are reported in the spreadsheet publicly available at https://goo.gl/aogGqs.

Table 1. Calculated metamodel clusters

(a) Cosine similarity

Thrsd	**#clst**	**Avg. clst size**	**Max clst size**	**#non-sing clst**	**Corr. with IM**
0.1	14	21.071	282	1	0.16
0.15	25	11.800	264	4	0.23
0.2	48	6.146	224	7	0.31
0.25	86	3.430	186	8	0.39
0.3	113	2.218	127	15	0.45
0.35	170	1.735	63	24	0.48
0.4	201	1.468	31	29	**0.49**
0.45	215	1.372	22	28	**0.49**
0.5	227	1.300	15	24	**0.49**
0.55	236	1.250	14	20	**0.49**
0.6	245	1.204	14	17	0.48
0.65	249	1.850	14	15	0.46
0.7	250	1.180	14	16	0.45
0.75	256	1.152	13	13	0.44
0.8	261	1.130	12	13	0.42
0.85	264	1.117	11	14	0.40
0.9	271	1.089	8	12	0.39
0.95	283	1.042	4	9	0.37

(b) Dice's coefficient

Thrsd	**#clst**	**Avg. clst size**	**Max clst size**	**#non-sing clst**	**Corr. with IM**
0.1	1	295	295	1	0.00
0.15	1	295	295	1	0.02
0.2	1	295	295	1	0.03
0.25	1	295	295	1	0.04
0.3	1	295	295	1	0.05
0.35	2	147 .5	294	1	0.06
0.4	2	147 .5	294	1	0.08
0.45	2	147 .5	294	1	0.10
0.5	7	42.143	289	1	0.14
0.55	17	17.353	279	1	0.20
0.6	55	5.364	230	7	0.28
0.65	146	2.021	91	25	0.41
0.7	207	1.425	47	22	0.48
0.75	230	1.283	14	24	**0.49**
0.8	242	1.219	14	23	0.47
0.85	254	1.161	14	16	0.44
0.9	263	1.122	8	15	0.41
0.95	273	1.081	8	12	0.38

(c) Match-based similarity

Thrsd	**#clst**	**Avg. clst size**	**Max clst size**	**#non-sing clst**	**Corr. with IM**
0.1	45	6.555	228	8	0.26
0.15	96	3.072	152	20	0.41
0.2	157	1.878	72	28	0.50
0.25	192	1.536	19	32	**0.52**
0.3	214	1.378	14	32	0.50
0.35	227	1.299	14	26	0.48
0.4	234	1.260	14	24	0.47
0.45	238	1.239	14	25	0.46
0.5	245	1.204	14	21	0.45
0.55	250	1.180	13	18	0.44
0.6	256	1.152	12	15	0.43
0.65	257	1.148	12	15	0.42
0.7	259	1.139	12	16	0.41
0.75	263	1.122	8	17	0.40
0.8	268	1.101	6	16	0.39
0.85	272	1.085	4	14	0.38
0.9	280	1.054	4	12	0.37
0.95	288	1.024	3	6	0.35

(d) Containment-based similarity

Thrsd	**#clst**	**Avg. clst size**	**Max clst size**	**#non-sing clst**	**Corr. with IM**
0.1	2	147.5	294	1	0.08
0.15	2	147.5	294	1	0.09
0.2	2	147.5	294	1	0.12
0.25	4	73.75	292	1	0.16
0.3	11	26.818	284	2	0.20
0.35	15	19.667	275	3	0.24
0.4	18	16.389	273	2	0.29
0.45	40	7.375	241	6	0.36
0.5	66	4.470	213	6	0.40
0.55	73	4.041	204	7	0.41
0.6	144	2.049	121	18	0.47
0.65	167	1.766	72	23	0.49
0.7	189	1.561	21	30	0.49
0.75	215	1.372	16	28	0.50
0.8	223	1.323	16	24	**0.51**
0.85	228	1.294	16	22	**0.51**
0.9	233	1.266	14	21	**0.51**
0.95	239	1.243	14	17	0.50

(e) Execution time

Cosine similarity	**Dice's coefficient**	**Match-based similarity**	**Containment-based similarity**
≈ 10min	≈5min	≈4 hours	

Evaluation Metrics. The manually defined clusters are used to generate the incidence matrix I defined as follows:

$$I(i,j) = \begin{cases} 1 & \text{if } mm_i \text{ and } mm_j \text{ are grouped in the same cluster} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

where $i, j = 1, \ldots, n$, and mm_i and mm_j are metamodels in the data set manually processed. Moreover, for each selected threshold and similarity function we binarized the calculated similarity matrix. The basic idea of binarization consists in the introduction of binary values associated to the similarity values. Each binary value is 1 if the numerical value to which it is associated has a value above a certain *threshold*, 0 otherwise. More formally, a similarity matrix S is binarized in $B(S)$ as follow:

$$B(S(i,j)) = \begin{cases} 1 & \text{if } Sim(mm_i, mm_j) \geq threshold \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where $i, j = 1, \ldots, n$, and mm_i and mm_j are metamodels in the corpus, and Sim is the considered similarity measure.

Inspired by the external validation technique discussed in [22], we have validated the metamodel clustering approach by measuring the similarity between the incidence matrix I, and the binarized similarity matrix $B(S)$ induced by the adopted similarity measure and threshold. Intuitively, the more similar the incidence matrix I and $B(S)$ are the better is the considered clustering technique. To this end we have applied the MATLAB `corr2` function[7] that returns the correlation coefficient in the range of -1.00 (perfect negative correlation) and +1.00 (perfect positive correlation) between two matrices or vectors of the same size. A correlation with value 0 indicates that the two considered elements are not correlated.

Data Analysis. The outcomes of the performed experiments are reported in Table 1. In particular, for each similarity function and threshold (denoted with *Thrsd* in the table) the following data are shown:

- *#clst*: the number of identified clusters;
- *Avg. clst size*: the average number of metamodels in each cluster;
- *Max clst size*: the number of metamodels in the bigger cluster;
- *#non-sing clst*: the number of clusters consisting of more than one metamodel;
- *Corr. with IM*: it indicates the correlation index between the automatically calculated clusters and the one manually identified.

It is worth noting that the data reported in the tables can be reproduced by interacting with the cluster visualizer component discussed in the previous section, which permits to select the similarity measure and the preferred similarity threshold. The graphical representation of the retrieved clusters is updated in real time accordingly.

According to the calculated correlations shown in the last column of Table 1(a–d), the match-based similarity with 0.25 as threshold value optimally approximates the metamodel clusters that we have manually defined. However, according to Table 1e the match-based similarity is also one of the most time consuming measure like the containment-based similarity one. The text-based similarity measures take less time than that required by the structural-based similarities. This result depends on the matching function of EMFCompare, that

[7] http://it.mathworks.com/help/images/ref/corr2.html.

is exponential and depends on the number of all the elements contained in all the possible metamodels couples. This is not the case of text-based similarities that instead are based on the textual distance between the vectors encoding as strings the metamodels being analysed.

6 Discussion

The main strengths of the approach proposed in this paper are related to the advantages of classifying metamodels in repositories in an automated way instead of using manual techniques that are strongly correlated to the maintainer background. This permits to relieve maintainers and contributors of the responsibility of annotating metamodels, which is beneficial for the accuracy of core functionalities such as searching and browsing. Nevertheless, the approach as proposed in this paper can be enhanced in different directions as discussed below.

Similarity Measure: The generation of metamodel clusters strongly depends on the adopted similarity measure. Depending on the purpose of the desired clustering, a corresponding measure has to be properly selected from existing ones in literature or even defined from scratch. As we have presented in Sect. 4, besides using two similarity measures that are commonly applied to text documents, we have developed two additional similarity measures (namely match-based and containment-based) that are model specific (in fact, they rely on the match models calculated by EMFCompare). However, even if we restrict our focus on metamodels, it is unlikely to define a similarity measure that meets the requirement of any modeler. This justifies the decision of conceiving an extensible proximity calculator that can be enriched by adding the implementation of further similarity measures that the user can then select and play with from the front-end of the application.

Performance: The complexity of hierarchical agglomerative algorithms is $O(n^2\ log\ n)$ [12] with n the number of elements in the considered data set. Consequently, for very large data sets the adoption of alternative algorithms is suggested. The hierarchical agglomerative clustering technique used in this work is only one of the many possible ones. Moreover, the developed supporting tool is agnostic of the clustering technique. It is important to remark that the developed system nightly updates all the proximity matrices in order to consider in the clustering calculation also newly added metamodels. Thus users do not experiment performance issues when playing with the cluster visualizer since all the required data are already pre-calculated.

Cluster Characterizations: In the current version of the approach each metamodel is assigned to exactly one cluster only. However, in some cases a given metamodel might belong to different clusters simultaneously. Moreover, by exploiting the content of the considered metamodels and the corresponding descriptions, when available, it can be possible to create cluster labels automatically. Supporting such characterizations represents a relevant improvement that we intent to investigate in the future.

Experimentation and Evaluation: The implementation of the approach has been applied by considering the metamodels available in the Ecore Zoo. However, to better assess the validity of the approach and of the optimal parameters presented in the previous section, and to obtain more extensive feedback, it is necessary to consider an extended data set consisting of additional metamodels that can be retrieved from further repositories like ReMoDD.

7 Related Work

Clustering techniques have been used in several applications including software and data comprehension, data migration, and reverse engineering.

In [23] authors use clustering techniques and Model-Driven Reverse Engineering principles for software comprehension. In particular, authors start by extracting data from source code for the input data matrix construction. For the code extraction, they consider the paragraph as the smallest atomic unit and their cluster analysis is based on the hypothesis that record fields existing in the same paragraphs can be grouped. For the data matrix the chosen distance of similarity for the cluster identification is the Euclidean distance. In [24] authors propose a software automatic categorization system called MUDABlue, based on Latent Semantic Analysis (LSA), which is a method to extract and represent the usage of words in texts by means of statistical computations. Similarly to our approach, [23,24] propose techniques able to automatically categorize set of similar objects. Whereas they focus on software comprehension, our approach specifically focuses on metamodels stored in repositories.

In [25] authors present a methodology for handling the problem of database migration. The approach uses semantic clustering to facilitate the translation of extended entity relationship schema into complex objects schema. They start from an Extended Entity Relationships (EER) schema to create a set of clustered schemata such that each clustered schema corresponds to a level of abstraction and grouping of the initial schema. By iteratively shrinking portions of EER diagram into complex entities, the approach creates a schema of entities by adding a layer of abstraction. The user can select a level of clustering to show components at some degree of detail exactly like we do in our approach.

In [26] authors present an approach to support the visualization of large-scale diagrams, which are decomposed into clusters of model elements. Graph clustering techniques are employed by defining the node similarity in terms of node distance. Differently to our approach, in [26] authors apply clustering techniques to create sub-models whereas in our approach we categorize metamodels without splitting their contents.

The work in [27] presents a technique, which is based on metamodeling, Petri nets, and Facets for the analysis and clustering of requirements diagrams. Intuitively, the approach is able to obtain the domain description in terms of the relations and dependencies of modeled services. Then the analysis and the clustering of requirements are automatically calculated accordingly. The work in [28] presents a semi-automatic technique for the construction of feature models based on requirements clustering. This approach automates the activities of feature identification and reorganization using clustering techniques. Starting from an existing system, the main idea is that tight-related individual functional requirements are clustered into features, and functional features are organized into an application feature model, which is then merged into a domain feature model. Differently to our approach, [27,28] are specifically conceived for the management of functional requirements, and clustering techniques are employed at model level.

8 Conclusion and Future Work

In this paper, we studied the problem of the automated categorization of metamodel repositories. The proposed approach adopted an agglomerative hierarchical clustering algorithm, which according to different similarity metrics (some of them specifically devised for metamodels) detects the application domains represented in an arbitrary repository. The effect of these similarity metrics is evaluated over a corpus of metamodels: while metamodel-specific similarities perform better at the price of a high execution-time, generic text-based similarities still offer acceptable accuracy with much better execution-time.

Future plans include a more systematic experimentation of the available similarity metrics to provide repository maintainers with tools, which can offer to the user a multi-dimensional browsing experience. In addition, we are interested in understanding how to use the variance between the incidence matrix and the experimental data as a feedback to improve similarity measures. We plan to experiment the application of clustering techniques also on other sets of modeling artifacts, like models and model transformations. In particular, we want to investigate to what extent it is possible to substitute model transformations that have source and/or target metamodels in common belonging to the same clusters. Positive results in such direction might be beneficial for enhancing the reuse of model transformations.

References

1. Schmidt, D.C.: Guest editor's introduction: model-driven engineering. Computer **39**, 25–31 (2006)
2. France, R.B., Bieman, J.M., Mandalaparty, S.P., Cheng, B.H.C., Jensen, A.: Repository for Model Driven Development (ReMoDD). In: Proceedings of 34th International Conference on Software Engineering (ICSE), pp. 1471–1472. IEEE (2012)
3. Hein, C., Ritter, T., Wagner, M.: Model-driven tool integration with modelbus. In: Workshop Future Trends of Model-Driven Development at International Conference on Enterprise Information Systems (ICEIS), pp. 50–52 (2009)
4. Karasneh, B., Chaudron, M.R.V.: Online Img2UML repository: an online repository for UML models. In: Proceedings of the 3rd International Workshop on Experiences and Empirical Studies in Software Modeling at MoDELS, pp. 61–66 (2013)
5. Koegel, M., Helming, J.: EMFStore: a model repository for EMF models. In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, ICSE 2010, pp. 307–308. ACM (2010)

6. Kutsche, R., Milanovic, N., Bauhoff, G., Baum, T., Cartsburg, M., Kumpe, D., Widiker, J.: BIZYCLE: model-based interoperability platform for software and data integration. In: Proceedings of MDTPI at ECMDA (2008)
7. Bislimovska, B., Bozzon, A., Brambilla, M., Fraternali, P.: Textual and content-based search in repositories of web application models. ACM Trans. Web **8**, 11:1–11:47 (2014)
8. Bourque, P., Dupuis, R., Abran, A., Moore, J.W., Tripp, L.L.: The guide to the software engineering body of knowledge. IEEE Softw. **16**, 35–44 (1999)
9. Anquetil, N., Fourrier, C., Lethbridge, T.C.: Experiments with clustering as a software remodularization method. In: Proceedings of the Sixth Working Confernce on Reverse Engineering, WCRE 1999, pp. 235–255. IEEE Computer Society (1999)
10. Beck, F., Diehl, S.: On the impact of software evolution on software clustering. Empirical Softw. Eng. **18**, 970–1004 (2012)
11. Vanya, A., Holland, L., Klusener, S., van de Laar, P., van Vliet, H.: Assessing software archives with evolutionary clusters. In: 16th International Conference on Program Comprehension, pp. 192–201. IEEE (2008)
12. Jain, A.K., Murty, M.N., Flynn, P.J.: Data clustering: a review. ACM Comput. Surv. (CSUR) **31**, 264–323 (1999)
13. Basciani, F., Di Rocco, J., Di Ruscio, D., Di Salle, A., Iovino, L., Pierantonio, A.: MDEForge: an extensible web-based modeling platform. In: Proceedings of CloudMDE at MoDELS, pp. 66–75(2014)
14. Berkhin, P.: A survey of clustering data mining techniques. In: Kogan, J., Nicholas, C., Teboulle, M. (eds.) Grouping Multidimensional Data, pp. 25–71. Springer, Heidleberg (2006)
15. Steinbach, M., Ertöz, L., Kumar, V.: The challenges of clustering high dimensional data. In: Wille, L.T. (ed.) New Directions in Statistical Physics, pp. 273–309. Springer, Heidelberg (2004)
16. Tan, P.N., Steinbach, M., Kumar, V.: Introduction to Data Mining. Pearson Education, London (2006). Chapter 8
17. Di Rocco, J., Di Ruscio, D., Iovino, L., Pierantonio, A.: Collaborative repositories in model-driven engineering. IEEE Softw. **32**(3), 28–34 (2015)
18. Gomes, C., Barroca, B., Amaral, V.: Classification of model transformation tools: pattern matching techniques. In: Dingel, J., Schulte, W., Ramos, I., Abrahão, S., Insfran, E. (eds.) MODELS 2014. LNCS, vol. 8767, pp. 619–635. Springer, Heidelberg (2014)
19. Di Rocco, J., Di Ruscio, D., Iovino, L., Pierantonio, A.: Mining correlations of ATL model transformation and metamodel metrics. In: Proceedings of the Seventh International Workshop on Modeling in Software Engineering, MiSE 2015 - ICSE, pp. 54–59. IEEE Press (2015)
20. Di Rocco, J., Di Ruscio, D., Iovino, L., Pierantonio, A.: Mining metrics for understanding metamodel characteristics. In: 6th International Workshop on Modeling in Software Engineering, MiSE 2014 - ICSE, Hyderabad, India, 2–3 June 2014, pp. 55–60 (2014)
21. Dice, L.R.: Measures of the amount of ecologic association between species. Ecology **26**, 297–302 (1945)
22. Halkidi, M., Batistakis, Y., Vazirgiannis, M.: On clustering validation techniques. J. Intell. Inf. Syst. **17**, 107–145 (2001)
23. El Beggar, O., Bousetta, B., Taoufiq, G.: Comparative study between clustering and model driven reverse engineering approaches. Lect. Notes Softw. Eng. **1**(2) (2013)

24. Kawaguchi, S., Garg, P.K., Matsushita, M., Inoue, K.: Mudablue: an automatic categorization system for open source repositories. J. Syst. Softw. **79**, 939–953 (2006)
25. Missaoui, R., Godin, R., Sahraoui, H.: Migrating to an object-oriented database using semantic clustering and transformation rules. Data Knowl. Eng. **27**, 97–113 (1998)
26. Strüber, D., Selter, M., Taentzer, G.: Tool support for clustering large meta-models. In: Proceedings of the Workshop on Scalability in Model Driven Engineering, BigMDE 2013 at STAF, pp. 7: 1–7: 4. ACM (2013)
27. Lopez, O., Laguna, M.A., Garcia, F.J.: Reuse based analysis and clustering of requirements diagrams. In: Eighth International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ02), pp. 71–82 (2002)
28. Chen, K., Zhang, W., Zhao, H., Mei, H.: An approach to constructing feature models based on requirements clustering. In: Proceedings of 13th IEEE International Conference on Requirements Engineering, pp. 31–40 (2005)