A Tool for Clustering Metamodel Repositories

Francesco Basciani, Davide Di Ruscio, Juri Di Rocco, Alfonso Pierantonio
DISIM - University of L'Aquila (Italy) - Email: {name.surname@univaq.it}
Ludovico Iovino
Gran Sasso Science Institute, L'Aquila, Italy - Email: {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 categorization mechanisms (if any) are based on manual activities. When dealing with large numbers of modeling artifacts, such limitations increase the effort related to both managing and reusing artifacts stored in model repositories. By focusing on metamodels management, in this paper we propose a clustering tool for automatically organizing stored metamodels and provide users with repository overviews as, for instance, the application domains covered by the available metamodels. The approach has been implemented and integrated in the MDEForge repository[1].**

I. Motivation and goals

The increasing adoption of Model-Driven Engineering (MDE) [19] in business organizations led to the need of gathering artifacts in model repositories [11]. Several model repositories (see [12], [13], [15], [16], [11] just to mention a few) have been introduced in the past decade. Among them *metamodel zoos* (as for instance the Ecore Zoo[2]) 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 [4]: an activity that is 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 [5]. In order to mitigate the difficulties related to the manual categorization of metamodels, we propose a clustering tool for metamodel repositories: an unsupervised procedure, which automatically organizes metamodels into clusters. Mutually similar artifacts are grouped together depending on a proximity measure, whose definition can be given according to specific search and browsing requirements. The tool is based on agglomerative hierarchical clustering [14] and explores well-known proximity measures as well as metamodel-specific ones, each providing different browsing characteristics.

[1]This research was supported by the EU through the Model- Based Social Learning for Public Administrations (Learn Pad) FP7 project (619583).

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

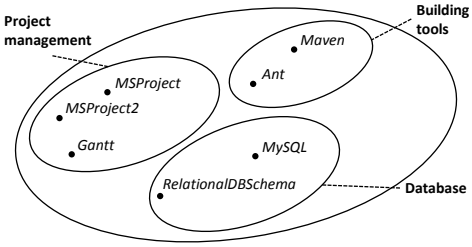

Fig. 1. Example of classified metamodels

II. Background

Even though several MDE approaches have been conceived over the last years to support a wide range of model management activities, model repositories are not yet as well developed and widespread as source-code repositories [7], [4]. Most of the potential benefits of the existing online 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 issues:

I1. 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 that 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;

I2. none of the available repositories provide mechanisms to automatically categorize the stored artifacts, thereby 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.

In the next sections we propose a tool able to address these challenges by focusing on the management of metamodels stored in publicly available repositories.

III. Proposed metamodel clustering approach

In order to deal with the issues discussed in Section II in this section we propose an unsupervised metamodel clustering mechanism that permits to automatically organize unstructured metamodel repositories and provide the users with overviews of the available metamodels.

A. Overview

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 representing the similarities of all the metamodels available in the repository, and then applying the clustering algorithm.

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 of a singleton cluster, i.e., a cluster consisting of only one element. The repository maintainer can periodically consider all the available singleton clusters and verify if they have been created, e.g., because of 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 to 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* previously discussed.

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 metamodels clusters and give also the means to navigate them and retrieve detailed information about their content if requested by the user.

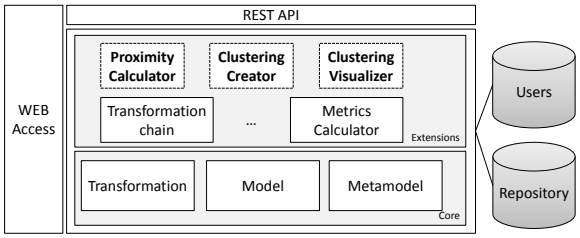

Fig. 2. MDEForge Architecture

B. Supporting tool

The proposed clustering method has been implemented as extensions of the MDEForge platform [1]. In particular, as shown in Fig. 2, 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. Both core service and extensions are available through Web access and programmatic interfaces (API) that enable their adoption as software as a service. For instance, in [2] 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 [8], [6]. In the remainder of the section, we give details about the extensions that are shown in dashed boxes in Fig. 2 and that we have developed to support the proposed metamodel clustering approach. Concerning the other services of MDEForge the reader can refer to [1], [7].

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. 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. [14]. 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 [3]. Among those typically applied to text documents we have considered the *cosine similarity* [3] and the *Dice's coefficient* [9] with the aim of relating the similarity of two metamodels on the terms used therein and consequently on the corresponding application domains. Moreover, we have developed two additional similarity functions specifically conceived for modeling artifacts. Both of them rely on the matching models calculated by means of EMFCompare[3]: i) *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

[3]http://www.eclipse.org/emf/compare/

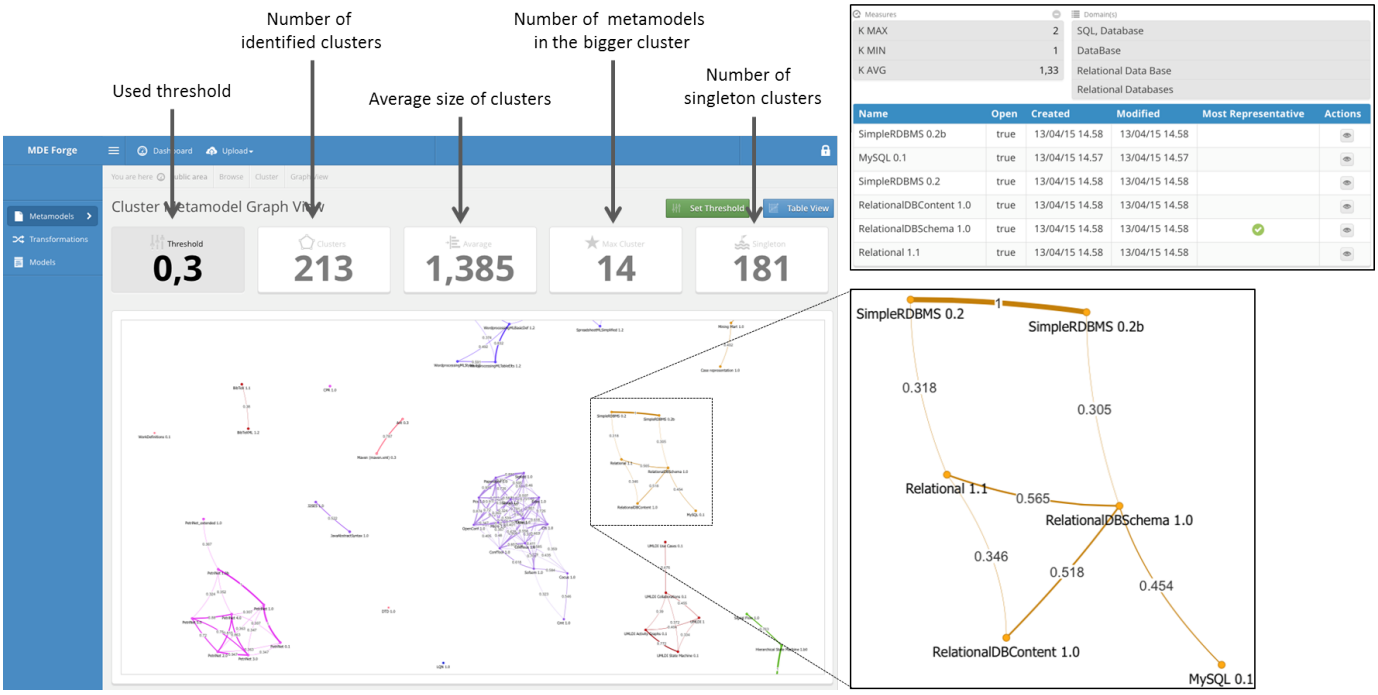

Fig. 3. Sample visualizations of automatically created metamodel clusters

of metamodels; ii) *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 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.

Clustering Creator: by using the proximity calculator previously discussed, it creates clusters of metamodels by applying the agglomerative hierarchical clustering algorithm. 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.

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. 3 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 each 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. 3. In particular, given a cluster all the contained metamodels are listed together with additional information like 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.

IV. Application 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. We have applied the clustering technique by using the four similarity functions mentioned in the previous section and by specifying different thresholds. Due to space constraints in this section we focus on the match-based similarity measure. For the same reason, the process that we have followed to validate the developed clustering technique is also omitted. It is worth noting that the data reported in Table I can be reproduced by interacting with the cluster visualizer component discussed in the previous section, which permits to select the similarity measure to be used and the desired similarity threshold. Then the graphical representation of the retrieved clusters is updated accordingly.

Figure 4 shows the number of clusters that are identified with respect to the chosen similarity threshold. A too low threshold correponds to consider the repository population almost undistinguished, whilst a too high threshold returns too many clusters with too few elements.

V. Related Work

Clustering techniques have been used in several applications including software and data comprehension. In [18] the authors presents a methodology for handling the problem of database migration. The approach uses semantic clustering to facilitate the translation of extended entity relationship schema into a schema of complex objects. They start from an Extended

Threshold	Clusters	Avg. cluster size	Max cluster size	Singleton clusters
0.1	45	6.555	228	37
0.15	96	3.072	152	76
0.2	157	1.878	72	129
0.25	192	1.536	19	160
0.3	214	1.378	14	182
0.35	227	1.299	14	201
0.4	234	1.260	14	210
0.45	238	1.239	14	213
0.5	245	1.204	14	224
0.55	250	1.180	13	232
0.6	256	1.152	12	241
0.65	257	1.148	12	242
0.7	259	1.139	12	243
0.75	263	1.122	8	246
0.8	268	1.101	6	252
0.85	272	1.085	4	258
0.9	280	1.054	4	268
0.95	288	1.024	3	282

TABLE I
MATCH-BASED SIMILARITY

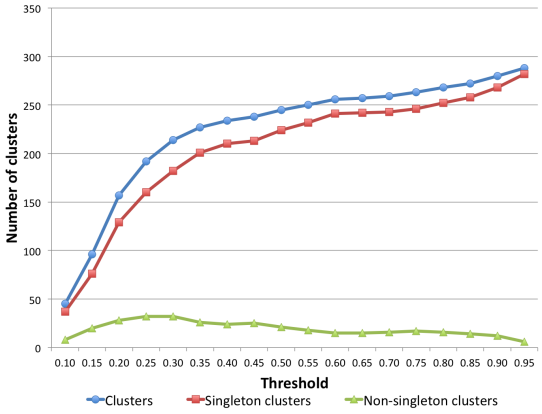

Fig. 4. Match-based Similarity thresholds

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 complex entities, hiding the details about the components. The user can select a level of clustering to show components at some degree of detail exaclty like we do in our approach. In [10] authors use clustering techniques and Model-Driven Reverse Engineering principles for software comprehension. In particular, the authors start by extracting data from source code for the input data matrix construction. In 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. The paper in [20] presents a tool for the decomposition of a metamodel into clusters of model elements. The authors claim that large-scale diagrams, representing models and metamodels, are often difficult to understand for the lack of appropriate modularization structures that allow examining a model in sub-parts. This work provides a meaningful way to split a monolithic model into sub-models for the comprehension and maintenance. The work in [17] 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.

VI. Additional Information

– MDEForge website and source code: http://www.mdeforge.org
– Related publications: [1], [2], [7], [8], [6]

References

[1] F. Basciani, J. Di Rocco, D. Di Ruscio, A. Di Salle, L. Iovino, and A. Pierantonio. MDEForge: an Extensible Web-Based Modeling Platform. In *Procs of CloudMDE@MoDELS 2014, Valencia, Spain, September 30, 2014.*, pages 66–75, 2014.

[2] F. Basciani, D. Di Ruscio, L. Iovino, and A. Pierantonio. Automated Chaining of Model Transformations with Incompatible Metamodels. In *Procs. of MODELS 2014*, pages 602–618, 2014.

[3] P. Berkhin. A survey of clustering data mining techniques. In J. Kogan, C. Nicholas, and M. Teboulle, editors, *Grouping Multidimensional Data*, pages 25–71. Springer Berlin Heidelberg, 2006.

[4] B. Bislimovska, A. Bozzon, M. Brambilla, and P. Fraternali. Textual and Content-Based Search in Repositories of Web Application Models. *ACM Transactions on the Web*, 8(2):1–47, Mar. 2014.

[5] P. Bourque, R. Dupuis, A. Abran, J. W. Moore, and L. L. Tripp. The Guide to the Software Engineering Body of Knowledge. *IEEE Software*, 16(6):35–44, 1999.

[6] J. Di Rocco, D. Di Ruscio, L. Iovino, and A. Pierantonio. Mining metrics for understanding metamodel characteristics. In *Procs. MiSE 2014 at ICSE 2014*, pages 55–60, 2014.

[7] J. Di Rocco, D. Di Ruscio, L. Iovino, and A. Pierantonio. Collaborative repositories in Model-Driven Engineering. *IEEE Software*, pages 28–34, May 2015.

[8] J. Di Rocco, D. Di Ruscio, L. Iovino, and A. Pierantonio. Mining Correlations of ATL Model Transformation and Metamodel Metrics. In *Procs of MiSE 2015 at ICSE 2015*, 2015.

[9] L. R. Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3):pp. 297–302, 1945.

[10] O. El Beggar, B. Bousetta, and G. Taoufiq. Comparative study between clustering and model driven reverse engineering approaches. *Lecture Notes on Software Engineering*, 1(2), 2013.

[11] R. B. France, J. M. Bieman, S. P. Mandalaparty, B. H. C. Cheng, and A. Jensen. Repository for Model Driven Development (ReMoDD). In *Procs. of ICSE 2012*, pages 1471–1472. IEEE, 2012.

[12] C. Hein, T. Ritter, and M. Wagner. Model-driven tool integration with ModelBus. *Workshop Future Trends of Model-Driven*, 2009.

[13] T. Holmes, U. Zdun, and S. Dustdar. Automating the Management and Versioning of Service Models at Runtime to Support Service Monitoring. In *EDOC*, pages 211–218, Sept. 2012.

[14] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM computing surveys (CSUR)*, 31(3):264–323, 1999.

[15] B. Karasneh and M. R. V. Chaudron. Online img2uml repository: An online repository for UML models. In *Procs of EESSMod 2013 at MoDELS 2013*, pages 61–66, 2013.

[16] R. Kutsche, N. Milanovic, G. Bauhoff, T. Baum, M. Cartsburg, D. Kumpe, and J. Widiker. BIZYCLE: Model-based Interoperability Platform for Software and Data Integration. In *Procs.of the MDTPI at ECMDA*, 2008.

[17] O. Lopez, M. A. Laguna, and F. J. Garcia. Reuse based analysis and clustering of requirements diagrams. In *Procs of REFSQ02*, pages 71–82, 2002.

[18] R. Missaoui, R. Godin, and H. Sahraoui. Migrating to an object-oriented database using semantic clustering and transformation rules. *Data and Knowledge Engineering*, 27(1):97 – 113, 1998.

[19] D. C. Schmidt. Guest Editor's Introduction: Model-Driven Engineering. *Computer*, 39(2):25–31, Feb. 2006.

[20] D. Strüber, M. Selter, and G. Taentzer. Tool support for clustering large meta-models. In *Procs. of BigMDE '13*, pages 7:1–7:4. ACM, 2013.