Tool Support for Clustering Large Meta-Models

Daniel Strüber, Matthias Selter, Gabriele Taentzer
Philipps-Universität Marburg
Fachbereich Mathematik und Informatik
Hans-Meerwein-Str., 35032 Marburg
{strueber, selterm, taentzer}@mathematik.uni-marburg.de

ABSTRACT

Ever-growing requirements, long-term evolution and modernization of software projects lead to meta-models of remarkable size, being difficult to comprehend and maintain. This paper presents a tool that supports the decomposition of a meta-model into clusters of model elements. Methods proposed in the research area of graph clustering, aiming at the desired properties of high cohesion and low coupling, have been integrated in the tool. The methods are customized not only to utilize the underlying graph structure, but also the semantic information given in meta-models. An evaluation of the tool is provided in terms of a case study.

Categories and Subject Descriptors

D.2.1 [**Sw. Engineering**]: Requirements / Specifications; D.2.8 [**Sw. Engineering**]: Distribution, Maintenance, and Enhancement

General Terms

Design, Algorithms

1. INTRODUCTION

Incrementally evolving requirements and inflating software evolution steps such as the compilation of models from multiple sources are reflected in monolithic meta-models of remarkable size. In software modernization projects, meta-models for systems obtained from reverse engineering techniques may be inscrutable at all due to their bulkiness. Crucial challenges to be dealt with facing such large models are model comprehension, synchronization in collaborative editing as well as time and memory efficiency of analysis performed on these models.

In terms of standardized meta-modeling frameworks such as the Eclipse Modeling Framework (EMF) [1], a meta-model is manifested by a tree-like structure of nested packages comprising sets of classes. To facilitate comprehension and convenient editing, diagrams corresponding for whole meta-models or sub-models can be automatically derived. Diagrams have a graph-like structure with classes as nodes and references or associations running between the classes as edges. Diagrams provide a convenient facility for the comprehension of small to medium-scale models. However, as for large-scale diagrams, comprehension is often hindered by the lack of appropriate modularization structures that allow examining a model in parts.

This paper is based on the claim that providing a meaningful splitting of a monolithic model into sub-models is beneficial for the comprehension and maintenance of the model. In the scope of this work, we define *meaningful* as *taking into account the structural information present in a meta-model*: On the one hand, cohesion and coupling. On the other hand, the semantic information provided by inheritance and containment relations.

The problem of decomposing a graph into sub-graphs with high cohesion and low coupling has been adressed by graph clustering. A plethora of graph clustering methods has been proposed and employed in domains such as database engineering, social, biological and information networks, and stock market analysis. An overview is provided in [2].

The main contribution of this paper is a tool that utilizes graph clustering in order to establish a meaningful grouping of meta-models into sets of sub-models. The individual parts of the contribution can be summarized as follows:

- We provide a tool for customizable clustering support for meta-models, comprising a graphical user interface and a clustering engine. An overview on the tool and its usage is provided in Sect. 2.
- We discuss how graph clustering can be made applicable to meta-models. This is the subject of Sect. 3.
- We evaluate the tool in terms of a small case study. Sect. 4 elaborates how the tool can be used to examine a sample model from the domain of adaptive multi-agent system engineering.

As the discussion of related work in Sect. 5 points out, graph clustering has been applied to models and meta-models in particular before. However, a configurable and user-oriented graph clustering tool has not been invested yet. Preliminary work on the subject of model splitting has been done in the context of a distributed modeling process [3]. While this former contribution provides a formalization for the result of splitting a model into a set of components, it does not describe a method of automation.

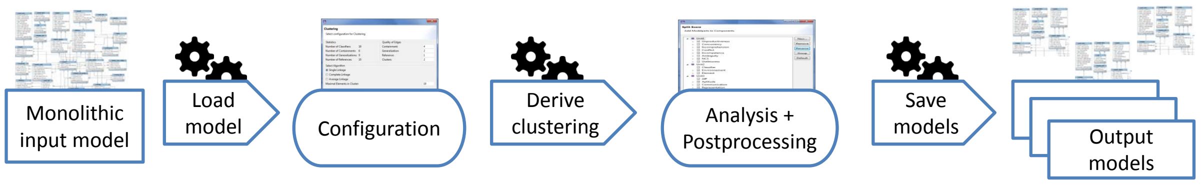


Figure 1: Tool workflow with artifacts (rectangles), automated activities (pentagons) and user activities (rounded rectangles)

2. TOOL OVERVIEW

The clustering tool takes a model as input, performs a clustering on its elements, displays the result and optionally allows saving the resulting sub-models. Its workflow is executed as shown in Fig. 1: At first, an input model is loaded. A wizard allows configuring the clustering according to the user's needs. It is possible to select the clustering algorithm being used and to adjust the weighting parameters for all edge kinds: reference, containment, and generalization. Then, the clustering is performed. The presentation of the result can either be used for analytical purposes (e.g., comprehension) or for saving the sub-models as interconnected individual models. For this use case, the wizard provides postprocessing capabilities such as naming the sub-models.

3. GRAPH CONVERSION

This section gives an insight on how the core component of the tool, the cluster derivation, works. The actual clustering is perfomed by selectable algorithms from graph clustering research [4] operating on a distance matrix of nodes. Hence, the tool converts a model into a distance matrix: Firstly, the model is converted into an undirected weighted graph. Secondly, distances between the nodes are computed.

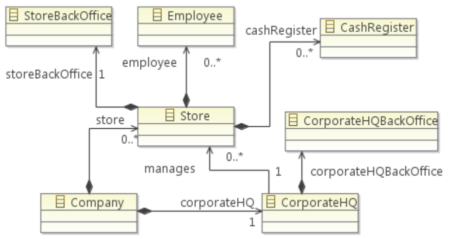


Figure 3: Class diagram of a Company Structure

3.1 Directed Graph with Edge Kinds

Fig. 3 shows a prototypical meta-model for a company management system with store and employee management capabilities. Our aim is to convert this model into a graph that can be used as input for graph clustering while preserving as much information from the model as possible.

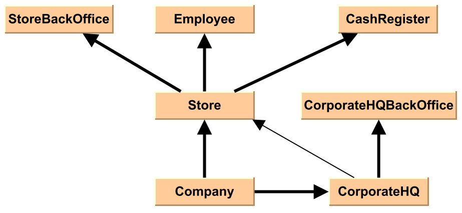


Figure 4: Directed Unweighted Graph

We first derive a directed graph with distinguished kinds of edges. Classes of the model become nodes in the graph. As for edges, in general terms, three different kinds occur: *containment*, *reference*, and *generalization*. The result of transforming the example model, shown in Fig. 4, contains the first two mentioned kinds only: Containment edges are painted bold, reference edges are painted thin.

3.2 Weighted Undirected and Distance Graph

The clustering algorithms employed by the tool are based on node similarity being expressed as node distance. Hence, a graph that provides node distances, i.e., a weighted graph, is required. Since similarity is a symmetric relation, we can neglect directionality. The first step to obtain such a graph is to assign a quality for each edge. The quality is a positive integer which depends on the edge kind. If there is more than one edge between two nodes, we sum up their qualities to combine them to one edge. Consequently, a high edge quality between two vertices indicates a close distance between the vertices. The second step is to derive a distance graph. Our distance function Δ inverts the quality of an edge by subtracting it from the highest quality present: $\Delta(e) = q(e_{maxq}) + 1 - q(e)$. The upper half of Fig. 5 shows the weighted undirected graph for the example. We chose edge qualities of 4 for containment and 2 for reference, taking containment as a strong type of reference into account. The lower half shows the corresponding distance graph.

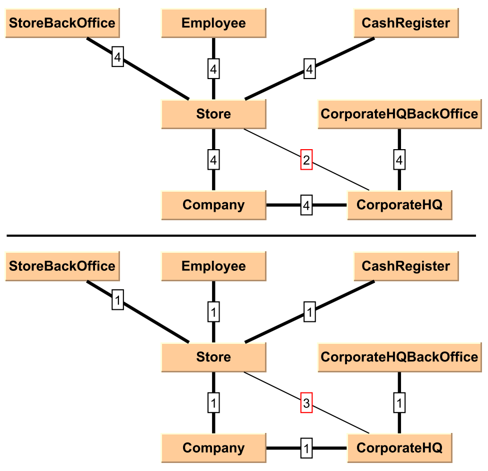


Figure 5: Weighted Undirected and Distance Graphs

3.3 Building a Distance Matrix

In order to build the distance matrix, we need to get the shortest pairwise distances between all vertices. The matrix can then be used as input for clustering algorithms.

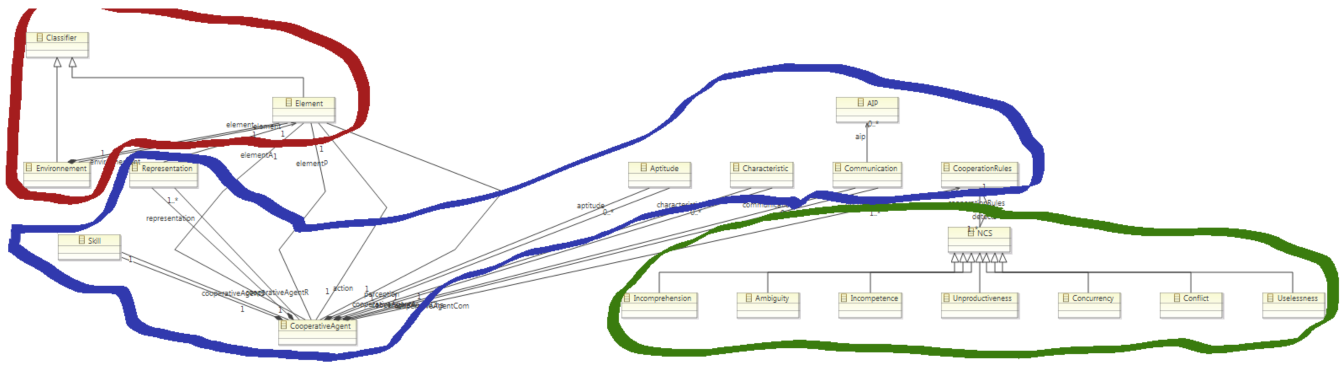


Figure 2: Result of clustering the ADELFE meta-model

$$
\begin{array}{c|ccccccc}
 & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
0 & 0 & 1 & 1 & 1 & 1 & 2 & 3 \\
1 & 1 & 0 & 2 & 2 & 2 & 3 & 4 \\
2 & 1 & 2 & 0 & 2 & 2 & 3 & 4 \\
3 & 1 & 2 & 2 & 0 & 2 & 3 & 4 \\
4 & 1 & 2 & 2 & 2 & 0 & 1 & 2 \\
5 & 2 & 3 & 3 & 3 & 1 & 0 & 1 \\
6 & 3 & 4 & 4 & 4 & 2 & 1 & 0
\end{array}
$$

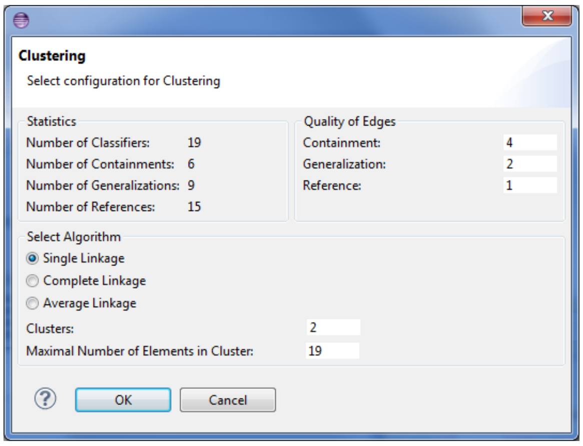
The matrix indexes correspond to classes as follows: 0: Store, 1: StoreBackOffice, 2: Employee, 3: CashRegister, 4: Company, 5: CorporateHQ, 6: CorporateHQBackOffice.

3.4 Application of a Clustering Algorithm

The implementation currently provides support for three hierarchical bottom-up clustering algorithms known for good performance: *Single Linkage, Complete Linkage* and *Average Linkage*. They work as follows: Initially, one cluster for each node is maintained. Then, a series of iterations is performed where two clusters are merged in each iteration. The algorithm terminates when a target number of clusters or an upper bound for the number of elements per cluster has been reached. The algorithms differ in the distance function employed for the comparison of clusters: Single Linkage in each step merges the two clusters maintaining the one pair of elements of overall minimal distance. Complete Linkage merges two clusters where the maximum pair-wise element distance is minimal. Average Linkage merges clusters based on minimal mean distance of elements. Further explanation is found in one of the excellent publications on graph clustering [4]. As our running example is too small to cluster usefully, an example is shown in the following section.

4. CASE STUDY

This section presents a small case study to evaluate how the clustering tool fulfills its functional requirements: (R1) the decomposition of a model into a set of sub-models with high coherence and low coupling; (R2) the ability to customize this decomposition based on generalization and containment properties. We chose a sample model large enough to contain some more and some less coherent parts, but small enough to be suited for presentation: ADELFE, shown in Fig. 2, is a meta-model devoted to the development of adaptive multi-agent systems [5].


Clustering
Select configuration for Clustering

Statistics
Number of Classifiers: 19
Number of Containments: 6
Number of Generalizations: 9
Number of References: 15

Quality of Edges
Containment: 4
Generalization: 2
Reference: 1

Select Algorithm
[✓ Single Linkage]
Complete Linkage
Average Linkage

Clusters: 2
Maximal Number of Elements in Cluster: 19

Figure 6: Configuration Dialog showing default values

4.1 Configuration

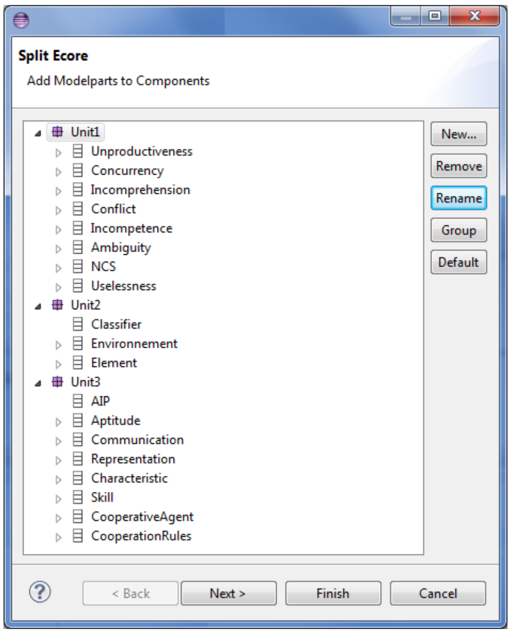
Fig. 6 shows the configuration dialog. Its upper left-hand part displays a set of metrics that have been applied on the input model: total numbers of classifiers, containments, generalizations, and references. Changing the quality parameters in the upper right part determines which parts of the model will be estimated as showing a high cohesion.

Our rationale for the assignment of the parameters is the following: Containment is a strong kind of reference and thus should always rate higher than plain references. Generalization, in contrast, is orthogonal to references. From the observation of multiple large meta-models, we witnessed that some meta-models, especially for technical domains, comprise a continuous generalization hierarchy while others do not. We assume that the information given by generalization is more meaningful if only a subset of model elements is subject to generalization. As suggested by the metrics, this is the case in ADELFE. Hence, we adjust the generalization quality from the default value of 2 to 6.

Further customizations available in the lower part are choice of algorithm, number of clusters, and maximal number of elements per cluster. Clustering algorithms have an inherent tradeoff between runtime and quality: *Single Linkage*, for instance, is known to be efficient, but likely to produce chains as it combines heterogenous clusters if they contain a pair of two closely related nodes. Limiting cluster size is useful to avoid large size differences between clusters.

4.2 Splitting Result

Fig. 7 shows the wizard view of resulting clusters, corresponding to the graphical view in Fig. 2. The clustering yields three clusters; two with eight elements and one with three elements. The cluster labeled as *Unit1* is recognizable as highly cohesive through inheritance. It comprises one superclass with seven children subclasses. *Unit2* comprises three classes which on the one hand have a strong cohesion through forming a containment hierarchy, on the other hand belong to the same inheritance structure. *Unit3* mainly includes classes sharing containment and ordinary references. For postprocessing, the user can use this wizard to add additional clusters or rename, remove, or regroup the existing ones. Please note that the implementation of the graphical view provided in Fig. 2 is a concern of future work.



Split Ecore

Add Modelparts to Components

```
Unit1
    Unproductiveness
    Concurrency
    Incomprehension
    Conflict
    Incompetence
    Ambiguity
    NCS
    Uselessness
Unit2
    Classifier
    Environnement
    Element
Unit3
    AIP
    Aptitude
    Communication
    Representation
    Characteristic
    Skill
    CooperativeAgent
    CooperationRules
```

Figure 7: Clustering Result and Postprocessing Dialog

5. RELATED WORK

A tool with functionality similar to the one described here has been deployed as part of the Democles project [6, 7]. The supported use cases are vastly the same as in our tool: comprehension, decomposition and pruning operations for a given model. However, in contrast to our approach, the Democles tool does not perform a clustering, but an analysis for strongly connected components. It cannot be configured towards component number and component size as supported by our tool. Furthermore, Democles does not utilize the information given by containment and generalization as it is not specialized on meta-models.

Streekmann and Hasselbring propose an architecture restructuring process involving graph clustering [8]. Their use case is to substitute a legacy system architecture with an improved architecture. A modeler defines the target architecture and, furthermore, an initial mapping of legacy architecture classes to target architecture components. Hierachical clustering is applied in order to assign the model elements neglected by the initial mapping. Our tool, in contrast, does not rely on the manual effort of defining an architecture and an initial mapping. The trade-off is that predefining these artifacts may give a more appropriate result.

Model comprehension is one of the main use cases of the large body of work that has been accomplished on model slicing, a survey being presented in [9]. The goal of model slicing is to extract sub-models from a model following some pre-defined slicing criteria. One use case for model slicing is to examine the relationships and dependencies of one particular class. Model slicing can be thought of as a complementary bottom-up approach to the top-down approach of model clustering: model clustering provides an outline view on a model in terms of an overview of distinct neighborhoods. Slicing then allows tracing one class of interest.

6. CONCLUSION

The contribution of this paper is a tool that allows performing a cluster analysis on a large meta-model. Its main use case is to improve model maintainability by facilitating model comprehension. For this sake, modularity is established by means of decomposing the input model to a set of sub-models. A concern of future work is not only to derive a set of sub-models, but also explicit export and import interfaces for the sub-models. The tool could then be integrated into the distributed modeling process proposed in [3]. Furthermore, splitting of model instances along the meta-model decomposition is to be investigated. Another concern of future work is to evaluate the tool on larger examples with significant cross-dependencies such as UML.

7. REFERENCES

[1] Eclipse Consortium. Eclipse Modeling Framework (EMF) – Version 2.5. `http://www.eclipse.org/emf`.

[2] Satu Elisa Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, 2007.

[3] Daniel Strüber, Gabriele Taentzer, Stefan Jurack, and Tim Schäfer. Towards a distributed modeling process based on composite models. In *Fundamental Approaches to Software Engineering*, pages 6–20. Springer, 2013.

[4] Trevor Hastie, Robert Tibshirani, and J Jerome H Friedman. *The elements of statistical learning*, volume 1. 2001.

[5] Adelfe. `http://www.irit.fr/ADELFE`.

[6] Democles. `http://democles.lassy.uni.lu`.

[7] Pierre Kelsen, Qin Ma, and Christian Glodt. Models within models: taming model complexity using the sub-model lattice. In *Fundamental Approaches to Software Engineering*, pages 171–185. Springer, 2011.

[8] Niels Streekmann and Wilhelm Hasselbring. Model-based architecture restructuring using graph clustering. In *Workshop Proceedings of the 13th European Conference on Software Maintenance and Reengineering*.

[9] Arnaud Blouin, Benoît Combemale, Benoit Baudry, and Olivier Beaudoux. Kompren: modeling and generating model slicers. *Software & Systems Modeling*, pages 1–17, 2012.