# Hierarchical Clustering of Metamodels for Comparative Analysis and Visualization

Önder Babur[1(✉)], Loek Cleophas[1,2], and Mark van den Brand[1]

[1] Eindhoven University of Technology, 5600 MB Eindhoven, The Netherlands
{O.Babur,L.G.W.A.Cleophas,M.G.J.v.d.Brand}@tue.nl
[2] Stellenbosch University, Matieland 7602, South Africa

**Abstract.** Many applications in Model-Driven Engineering involve processing multiple models or metamodels. A good example is the comparison and merging of metamodel variants into a common metamodel in domain model recovery. Although there are many sophisticated techniques to process the input dataset, little attention has been given to the initial data analysis, visualization and filtering activities. These are hard to ignore especially in the case of a large dataset, possibly with outliers and sub-groupings. In this paper we present a generic approach for metamodel comparison, analysis and visualization as an exploratory first step for domain model recovery. We propose representing metamodels in a vector space model, and applying hierarchical clustering techniques to compare and visualize them as a tree structure. We demonstrate our approach on two Ecore datasets: a collection of 50 state machine metamodels extracted from GitHub as top search results; and ∼100 metamodels from 16 different domains, obtained from AtlanMod Metamodel Zoo.

**Keywords:** Model-Driven Engineering · Model comparison · Vector space model · R · Hierarchical clustering

## 1 Introduction

Model-Driven Engineering (MDE) promotes the use of models and metamodels as first-class artefacts to tackle the complexity of software systems [15]. As MDE is applied for larger problems, the complexity, size and variety of models increase. With respect to model size, the issue of scalability has been pointed out by Kolovos et al. [15]. However, scalability with respect to model variety and multiplicity (i.e. dealing with a large number of different models) is also an important issue, and has been diagnosed by Klint et al. as an interesting aspect to explore [14]. There are many approaches to fundamental operations such as model comparison [25] and matching [16]; applied to problems such as model merging [8], versioning [3] and clone detection [9]; however those mainly focus

The research leading to these results has been funded by EU programme FP7-NMP-2013-SMALL-7 under grant agreement number 604279 (MMP).


A. Wąsowski and H. Lönn (Eds.): ECMFA 2016, LNCS 9764, pp. 3–18, 2016.
DOI: 10.1007/978-3-319-42061-5_1

on pairwise and 'deep' comparison of models to achieve high accuracy for a very small number of models. [23] further discusses the inadequacy of pairwise comparison for multiple models and proposes an N-way model merging algorithm.

Indeed, many problems in MDE involve processing a potentially large number of models. Some good examples are domain model recovery from several candidate (meta-)models [14], metamodel recovery [13] and family mining for Software Product Lines (SPL) from model variants [11]. A further problem can be given in the context of our ongoing project for a flexible multiphysics engineering simulation framework, where the domain contains an overwhelming number of tools [5], making it difficult to extend manual model extraction efforts such as in [6] to cover the whole domain.

We are interested in the case where a common (meta-)model is reverse engineered out of several candidate (meta-)models. For this paper we focus particularly on metamodel comparison and clustering; however our techniques are generic and thus applicable for the general model comparison and clustering problems. In essence, we treat metamodels as instances of the meta-metamodel. Having said that, the rest of the paper uses this convention. We argue that, as the number and variety of input metamodels gets larger, the initial data analysis and preprocessing step gets more and more relevant and necessary. This in turn calls for a need to inspect the dataset for an overview, identify potential relations between them such as proximities, cluster formations, outliers, etc. This information can be used potentially for filtering noisy data, for grouping metamodels, or even for determining the order of processing for a pairwise metamodel merging or SPL generation algorithm (see [23] for a discussion on how pairwise comparison order affects the outcome of merging multiple models).

In this paper, we present a continuation of our previous study [4]. We propose hierarchical clustering for comparative analysis and visualization of the dataset as a first explorative step in domain model recovery. We apply techniques from the Information Retrieval (IR) and unsupervised machine learning domains in the MDE context. In IR, a vector space model (VSM) is used to represent text documents, with vector elements corresponding to word occurrence (incidence) or frequency. We borrow this concept to represent metamodels as vectors of the unigrams from metamodel element identifiers. We apply an array of NLP techniques and weighting schemes to further improve the VSM and reduce the metamodel comparison problem into distance calculation between points in the vector space. We then use the R statistical software [20] to hierarchically cluster, analyse and visualize the dataset as a hierarchical structure. We demonstrate our approach on two Ecore datasets: a collection of 50 state machine metamodels extracted from GitHub as top search results, and ∼100 metamodels from 16 different domains, obtained from AtlanMod Metamodel Zoo.

*Objectives.* The purpose of this study is to answer the following questions:

- **RQ1.** How can we represent metamodels for large-scale comparative analysis?
- **RQ2.** How can we analyse, compare and visualize a large set of metamodels?

## 2 Preliminaries: Information Retrieval and Clustering

Information Retrieval [19] has a long history of developments in dealing with effectively indexing, analyzing and searching various forms of content including natural language text documents. As a first step for document retrieval in general, documents are collected and indexed via some unit of representation. Index construction can be implemented using models ranging from boolean indices to complex neural networks. One such model is the vector space model (VSM) with the following major components:

- A vector representation of (binary) occurrence of the vocabulary in a document, named *term incidence*;
- Optionally *zones* (e.g. 'author' or 'title' zones separate from the text bodies),
- Optionally weighting schemes to be used as multipliers such as:
  - *inverse document frequency (idf)* (see Sect. 3.1) to increase the discriminative effect of rare words,
  - zone weights, e.g. higher for important zones,
- Optionally Natural Language processing (NLP) techniques such as:
  - methods for handling compound terms, e.g. tokenization or multi-word similarity measures,
  - methods for detecting synonyms, hyponyms, and semantically related words, e.g. use of a stemmer or WordNet[1].

The VSM allows transforming each document into an $n$-dimensional vector, thus resulting in an $m \times n$ matrix where $m$ is the number of documents and $n$ is the size of the vocabulary.

Once the VSM is constructed, the similarity of documents can be defined as the distance between these vectors. There exist several distance a.k.a. similarity measures, such as Euclidian, Cosine or Manhattan, to be chosen considering the underlying problem domain and dataset. VSM with a selected distance measure is the prerequisite for identifying similar groups of documents in the vector space. This unsupervised machine learning technique is called clustering. Among many different clustering methods [12,19], there is a major distinction between flat clustering and hierarchical clustering. Flat clustering needs a pre-specified number of clusters and results in a flat assignment of each document into one cluster. Hierarchical clustering, on the other hand, does not require a pre-specified number of clusters, and outputs a hierarchy of proximities; it thus is more flexible and informative than flat clustering. Specifically, hierarchical agglomerative clustering (HAC) outputs a nested tree structure called *dendrogram*, which is suitable for visualization and manual inspection. As can be seen in Fig. 3, the leaves of the dendrogram represent data points, and each merge is represented by a horizontal line. The height of the merge point corresponds to the distance (inverse similarity) of the data points and/or subclusters.

The HAC algorithm calculates the pairwise distances of all the points in the dataset. In a bottom-up manner, it starts with each data point in a separate

[1] https://wordnet.princeton.edu/.

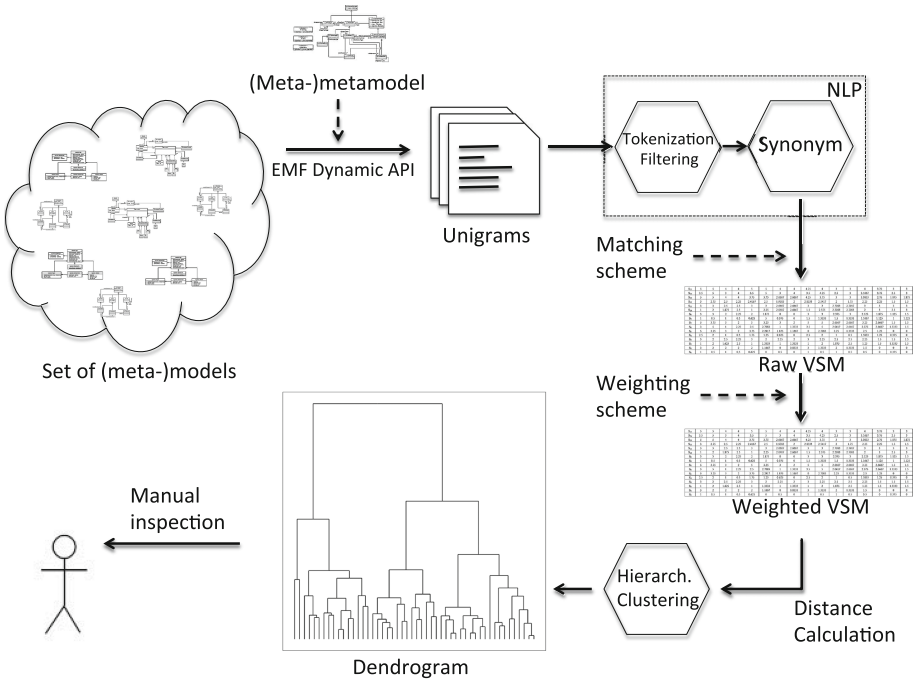



**Fig. 1.** Overview of our approach.

cluster and recursively merges similar points/clusters into bigger clusters. There is a further parameter for HAC for determining how this merge is decided with respect to the inter-cluster distance: single-link assumes cluster distance is the maximum similarity of any individual points in two clusters, while complete and average-link are the correspondingly minimum and average similarity.

## 3 Method for Metamodel Clustering

In this section, we elaborate our approach on a small example. The method is based on that in [4], extended with various features. We put emphasis on NLP aspects and real datasets, and as a result choose to use unigrams (item 2(a) below) rather than bigrams (sequence of two related unigrams, e.g. a class X with its attribute Y). The readers are referred to Sect. 5.1 for a short discussion on this choice. An overview of our approach is given in Fig. 1 with the main steps as:

1. Obtaining input dataset
   (a) Obtaining a set of metamodels with the same type, e.g. Ecore metamodels in our case, to be analyzed,
2. Creating VSM representation

   (a) Generating the unigram vocabulary (i.e. the element identifiers) from the input metamodels and the unigram types (similar to zones in IR) from the meta-metamodel (the generic Ecore meta-metamodel in our case, rather than lower level domain-specific ones),
   (b) Expanding the unigrams with tokenization, and then filtering e.g. stop-words,
   (c) Detecting synonyms and relatedness amongst tokens,
   (d) Utilizing a synonym and type matching mechanism/threshold,
   (e) Utilizing an idf and type-based weighting scheme,
   (f) Calculating the term incidence matrix,
3. Clustering
   (a) Picking a distance measure and calculating the vector distances,
   (b) Applying hierarchical clustering over the VSM,
   (c) Visualizing the resulting dendrogram for manual inspection.

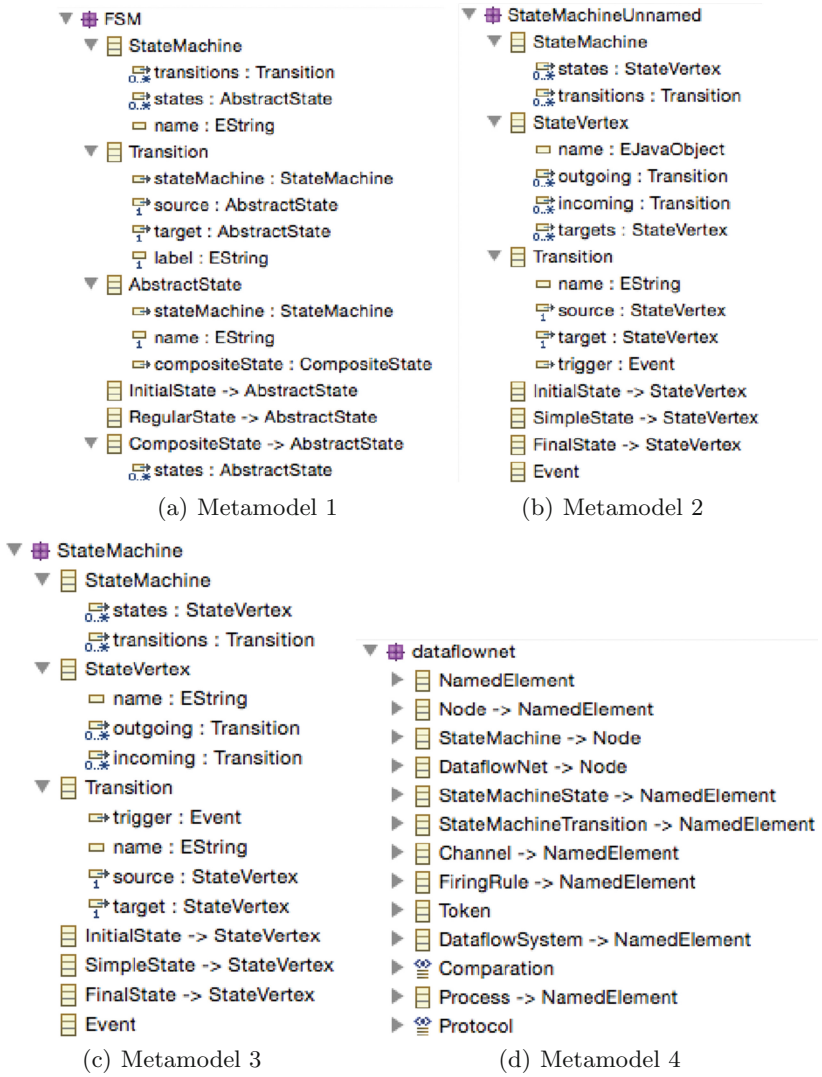
*A Small Example Dataset.* Here we introduce a small dataset of Ecore-based metamodels. In contrast to [4], we build this work directly on Ecore, though we extract a subset of the metamodel elements (see Sect. 3.1). Here we gather 4 metamodels related to state machines, selected from our first case study (Sect. 4.1). The dataset, depicted in Fig. 2, consists of two plain finite state machine (FSM) metamodels; one hierarchical FSM metamodel (the latter has the package name FSM though); and one data flow metamodel.

### 3.1 Representation as VSM

*Generating the unigram vocabulary.* From the input metamodels and Ecore meta-metamodel, we construct a typed unigram vocabulary. We adopt a *bag of words* representation for the vocabulary, where each item in the vocabulary is considered individually, discarding the context and order. The type information comes from Ecore ENamedElements, i.e. identifiers: we get the set {**EPackage, EDataType, EClass, EAttribute, EReference, EEnum, EEnumLiteral** and **EDataType**}. Next, we use the EMF Reflexive API (in Java) to recursively go over all the content for each metamodel element to extract the union of unigrams. The first metamodel in Fig. 2 would yield **Metamodel 1** = {**FSM(EPackage), StateMachine(EClass), transitions(EReference), states(EReference), name(EAttribute), ...**}. Note that several parts of Ecore are deliberately not included in the unigram generation such as EAnnotations and OCL constraints. These are negligible in our case studies, might require further techniques, and are left as future work.

*Vocabulary expansion with tokenization, and then filtering.* As identifiers in the metamodels typically are compound names (similar to source code identifiers), we apply tokenization and turn compound names into their tokens to include the vocabulary. We use the Identifier Name Tokenization Tool[2] for implementing this

[2] https://github.com/sjbutler/intt.




(a) Metamodel 1 (b) Metamodel 2

(c) Metamodel 3 (d) Metamodel 4

**Fig. 2.** Example dataset.

functionality. The types of the original identifiers are retained in the tokens. The expansion of **StateMachine(EClass)** for instance would yield **State(EClass)** and **Machine(EClass)** unigrams. Afterwards we apply a set of filters for the tokens: removal of stop words such as *'of'* and *'from'*, removal of overly short tokens ($< 3$ characters) and ones consisting of only digits. Note that having done this tokenization step, we use term and token interchangeably for this paper. It is also noteworthy to mention that tokenization reduces the vector space for large datasets significantly: e.g. from 7507 to 5842 for case study 2 (Sect. 4.2).

This contributes to the scalability of the approach with respect to the growing size of the dataset.

*NLP techniques for synonym and relatedness detection.* For the synonym and relatedness detection, we use another array of techniques after normalizing all the tokens into lower case. First of all, we use a Porter Stemmer (Java implementation[3]) for comparing word stems (e.g. *'located'*, *'location'* and *'locations'* have the common stem *'locat'* and therefore are considered synonyms. Next we measure the normalized Levenshtein distances of the tokens, and consider close words ($< 0.1$ difference) as synonyms. This allows for approximate string matching, tackling e.g. small typos. Finally, tokens which have a WordNet[4] WuP similarity score above a certain threshold (0.8 for the examples here) are considered synonyms. We use the WS4J Java library[5] for this calculation.

*Unigram matching scheme.* We further use a type matching and synonym matching scheme. When comparing two typed unigrams, we add a reducing multiplier of 0.5 for non-exact type matches and use the similarity score as a reducing multiplier for synonym matching. As an example a typed unigram **name(EAttribute)** would yield 1 when matched against itself, while yielding $0.5 * 0.88 = 0.44$ against **label(EReference)**, where 0.88 is the WordNet WuP similarity score of *'name'* and *'label'*. As mentioned before, a detailed evaluation of different values and parameter settings is out of scope for this paper and left as future work.

*Idf and type weighting scheme.* The similarity calculation described above gives a score in the range [0, 1] for each metamodel-token pair. On top of this, we apply a weighting scheme on the term incidence matrix, which includes two multipliers: an inverse document frequency (idf) and a type (zone) weight. The idf of a term $t$ is used to assign greater weight to rare terms across metamodels. Idf as the normalized log is defined as:

$$idf(t) = log_{10}\left(1 + \frac{\#\text{ total metamodels}}{\#\text{ metamodels with the term t}}\right) \tag{1}$$

Furthermore, a type weight is given to the unigrams representing their semantic importance. We use a similar scheme as in [4], this time for all the Ecore ENamedElements listed above. We claim, for instance, that classes are semantically more important than attributes, thus deserve a greater weight. We have used this experimental scheme for this paper:

$$\begin{aligned} typeWeight(t, w) : \{&\text{EPackage} \rightarrow 1.0, \text{EDataType} \rightarrow 0.2, \text{EClass} \rightarrow 1.0, \\ &\text{EReference} \rightarrow 0.5, \text{EAttribute} \rightarrow 0.3, \text{EEnum} \rightarrow 1.0, \\ &\text{EEnumLiteral} \rightarrow 1.0, \text{EOperation} \rightarrow 0.5, \text{EParameter} \rightarrow 0.1\} \end{aligned}$$

[3] http://tartarus.org/martin/PorterStemmer/.
[4] https://wordnet.princeton.edu/.
[5] https://github.com/coriane/ws4j.

A part of the resulting matrix where all the preprocessing steps above have been done, and the term incidences have been multiplied by idf and weights, is given in Table 1.

**Table 1.** Idf and type weighted term incidence matrix.

| Metamodel | FSM | State | Machine | source | label | Initial | Channels | ... |
|---|---|---|---|---|---|---|---|---|
| M1 | 0.35 | 0.15 | 0.15 | 0.09 | 0.05 | 0.15 | 0 | ... |
| M2 | 0 | 0.15 | 0.15 | 0.09 | 0.05 | 0.15 | 0 | ... |
| M3 | 0 | 0.15 | 0.15 | 0.09 | 0.04 | 0.15 | 0 | ... |
| M4 | 0 | 0.15 | 0.15 | 0 | 0.04 | 0.15 | 0.18 | ... |

### 3.2 Clustering

*Picking a distance measure and calculating the distance matrix.* As the next step of our approach, we reduce the metamodel similarity problem into a distance measurement of the corresponding vector representations of metamodels. We had previously suggested to pick Manhattan distance [4]. In common natural language text retrieval problems however, cosine distance is used most frequently. Based on the empirical comparisons between the two and the fact that cosine distance is a length normalized metric in the range [0, 1] (while Manhattan is not), we choose to use cosine distance for our current work. A quantitative evaluation of the various framework parameters such as distance measure and their effect on clustering is left as future work. $p$ and $q$ being two vectors of $n$ dimensions, cosine distance is defined as:

$$cosineDistance(p,q) = 1 - \frac{p \cdot q}{\| p \| \| q \|} = 1 - \frac{\sum_{i=1}^{n} p_i q_i}{\sqrt{\sum_{i=1}^{n} p_i^2} \sqrt{\sum_{i=1}^{n} q_i^2}}. \quad (2)$$

To be used by the hierarchical clustering, we calculate the pairwise distance matrix of all the models. The distance matrix for the example dataset is given in Table 2. We use the `lsa` package in R for this computation [27].

*Hierarchical clustering and visualization.* We apply agglomerative hierarchical clustering over the VSM to obtain a dendrogram visualization. We used the `hclust` function in the `stats` package [20] with average linkage to compute the dendrogram. The interpretation of this diagram depicted in Fig. 3 is as follows: the red and green dotted line at heights 0.3 and 0.6 (manually inserted by us) denote horizontal cuts in the dendrogram. Metamodel 4, which stays far above the cut, can be considered as a clear outlier. Depending on the requirements and interpretation of the user, Metamodels 1–3 can be considered to be in one single cluster (i.e. dendrogram cut at height = 0.6) or just Metamodels 2 and 3 (i.e. cut at height = 0.3).

**Table 2.** Pairwise distance matrix.

| | M1 | M2 | M3 |
|---|---|---|---|
| M2 | 0.61 | | |
| M3 | 0.56 | 0.10 | |
| M4 | 0.72 | 0.81 | 0.79 |

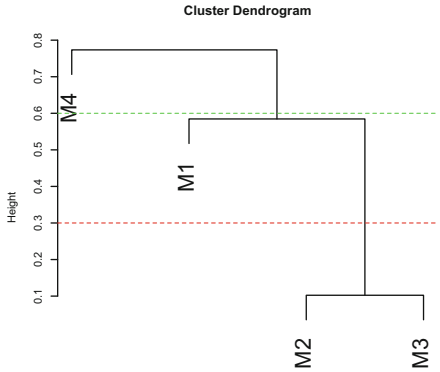



**Fig. 3.** Dendrogram of the examples. (Color figure online)

## 4 Case Studies

We introduce two case studies to demonstrate the feasibility of our approach.

### 4.1 Case Study 1 - GitHub Search Results

*Dataset design.* For this case study, we searched GitHub[6] on 11.02.2016 for Ecore metamodels using the search terms *'state machine extension:ecore'* and extracted the top 50 results out of 1089 (code) results in total, sorted by *Best Match* criteria. The search facility of GitHub has an internal mechanism for indexing and retrieving relevant text files. Although the intention of this search is to obtain various types of state machine metamodels, we expect to get a heterogeneous dataset, and apply clustering to give an overview of the results.

*Objectives.* This case study aims to demonstrate the applicability of our approach in a large dataset of a single domain (i.e. state machines), with possible duplicates, outliers, and subdomains. We are eventually interested in large (i.e. $> 3$ data points) groups of closely similar (e.g. cosine distance $< 0.8$) metamodels and wish to exclude the outliers. The fact that the we obtain metamodels through searching in GitHub also leads to a secondary objective of metamodel searching and exploration (e.g. for reuse, in the sense of traversing a repository/search results and finding the desired metamodels).

*Results.* Figure 4 shows the resulting dendrogram. We have visually identified and labelled the clusters from 1 to 5. Cluster 1 composes of two very similar (distance $< 0.1$) groups of duplicate metamodels (distance $= 0$) as basic FSMs with states, transitions and associations. In Cluster 2, there are two groups of UML-labelled metamodels with controller elements, triggers, etc. Cluster 3 has metamodels with specializations such as initial and final states, while Cluster 4

[6] https://github.com.

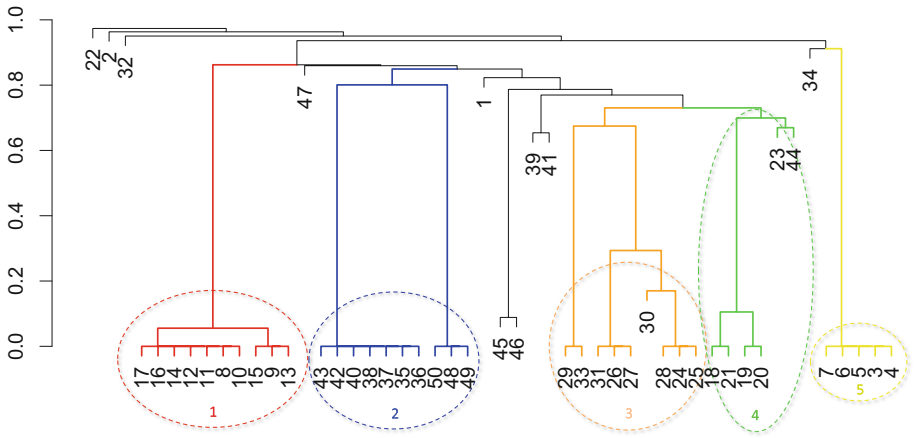



**Fig. 4.** Dendrogram of the first dataset.

has hierarchical state machines with composite states (Metamodel 23 is a false positive: it is labelled NHSM - *non*-hierarchical and is yet put in this cluster). Cluster 5 has duplicate metamodels labelled as AUIML with agents, messages, etc. and is clearly separate from the rest of the clusters. Outliers include a metamodel with identifiers in French (22), a train behaviour metamodel (2), the dataflow metamodel as given in the example dataset (34) and so on. The models 45, 46, 39 and 41 are deliberately not considered as a cluster due to the requirements we set above regarding cluster size and maximum distance.

### 4.2 Case Study 2 - AtlanMod Metamodel Zoo

*Dataset design.* For this case study, we used a subset of the Ecore metamodels in the AtlanMod Ecore Metamodel Zoo[7]. The Zoo is a collaborative open repository of metamodels in various formalisms including Ecore, intended to be used as experimental material by the MDE community. The repository itself has a wide range of metamodels from different domains; e.g. huge metamodels for programming languages or small class diagram examples for specific problems. We manually selected a subset of 107 metamodels, from 16 different domains. The domain labels are mostly retained as labelled in the repository. Table 3 depicts the domain decomposition. The cell below each domain shows the total number of metamodels in that domain, and the corresponding identifiers used in the resulting dendrogram in Fig. 5.

*Objectives.* This case study aims to demonstrate the applicability of our approach in a large dataset of multiple domains and subdomains. The domains are chosen to be in a wide range, hence the clustering is meant to show the groups

[7] http://web.emn.fr/x-info/atlanmod/index.php?title=Ecore.

**Table 3.** Number of metamodels in each domain in case study 2

| Bibliography | Conference | Business process | Bug tracker | Multi-agent | ADL |
|---|---|---|---|---|---|
| 8(1–8) | 14(9–22) | 6(23–28) | 3(29–31) | 2(32–33) | 15(34–48) |
| Build Tool | Data Warehouse | Database | Office | Performance | SBVR |
| 5(49–53) | 6(54–59) | 5(60–64) | 10(65–74) | 3(75–77) | 4(78–81) |
| Soft. Process | State Machine | Petri Net | Use Case | Total | |
| 3(82–84) | 8(85–92) | 11(93–103) | 4(104–107) | 107 | |

and subgroups in the dataset in a bird's eye point of view. The fact that the metamodels reside in a well-known repository also leads to a side-objective of model repository management and exploration.

*Results.* Figure 5 shows the resulting dendrogram. We have visually identified and labelled the clusters from 1 to 16. Let us summarize a part of this dendrogram. Cluster 1 (multi-agent) is recognizable as a separate small cluster from the rest of the dataset. Clusters 2 (petri nets) and 3 (state machines) reside as sibling branches. Similarly, clusters 4 (bibliography) and 5 (conference) are clearly detectable as sibling clusters. Cluster 6 and to some extent 8 are a mixture of individual metamodels from different domains, therefore are erroneous according to our initial categorization. Cluster 7 is of build tools. Cluster 9 (database) is in close proximity to the big cluster 10 (office), the latter of which can be decomposed into two subclusters (left subtree as Word, and right as Excel). Clusters 11–16 correspond to various remaining domains with varying percentages of false positives.

As an external measure of cluster validity, we employ the $F_{0.5}$ measure. Given $k$ as the cluster labels found by our algorithm, $l$ as the reference cluster labels and *cluster pairs* as the pairs of data points in the same cluster, $F_{0.5}$ can be defined as:

$$F_{0.5}(k,l) = \frac{1.25 * Precision(k,l) * Recall(k,l)}{0.25 * Precision(k,l) + Recall(k,l)} \tag{3}$$

$$Precision(k,l) = \frac{|\text{ cluster pairs in } k \cap \text{cluster pairs in } l|}{|\text{ cluster pairs in } k|} \tag{4}$$

$$Recall(k,l) = \frac{|\text{ cluster pairs in } k \cap \text{cluster pairs in } l|}{|\text{ cluster pairs in } l|} \tag{5}$$

The reason for selecting this measure is that the $F_\beta$ measure is more common than e.g. purity or the Rand index in the software engineering community, and that we value precision higher than recall; hence the $F_{0.5}$ variant. According to this formula, and using the R package `clusteval` [21] for the co-membership table computation, we obtain an $F_{0.5}$ score of 0.73 for our manual clustering.

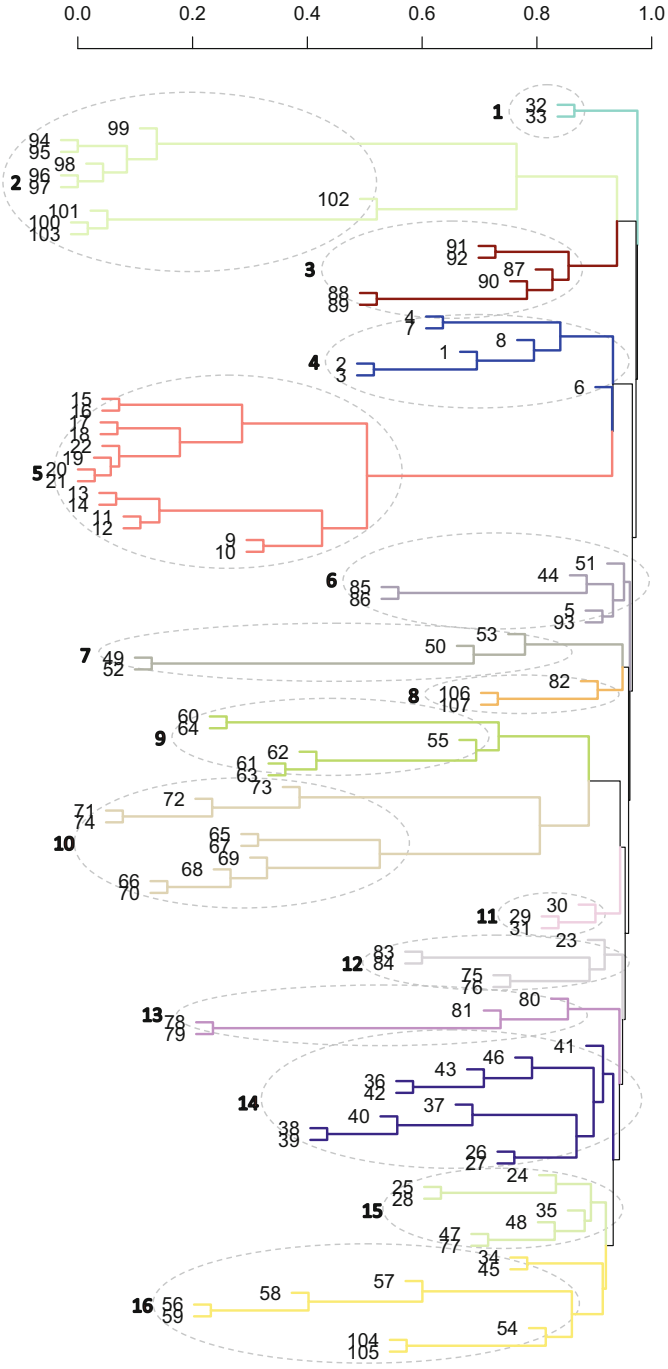



**Fig. 5.** Dendrogram of the second dataset.

## 5 Discussion

This paper improves our previous work in [4] considerably, in terms of NLP features and case studies on real datasets. Based on the two case studies, we confirm our previous claim that a statistical perspective on the comparative analysis and visualization of large datasets seems promising. We make a step towards the handling of large datasets. Using VSM allows a uniform representation of metamodels for statistical analysis, while the accompanying idf and type-based weighting scheme yields a suitable scaling in the vector space (**RQ1**). Using a distance measure and hierarchical clustering over VSM, many characteristics and relations among the metamodels, such as clusters, subclusters and outliers, can be analyzed and visualized via a dendrogram (**RQ2**).

Particularly for the first case study, it is clearly noticeable that there are distinct outliers and groupings in the search results. This information can be used for instance by a domain model recovery tool to improve the quality of the domain model. Furthermore, the model search functionality, either in GitHub or a specialized model search engine such as [18], can improve the navigation or precision of the search results. The second case study, on the other hand, deals with a heterogeneous set of domains and allows identifying domains, subdomains and also the proximities between related ones. We achieve a $F_{0.5}$ score of 0.73 from our manual clustering, which can be considered quite high for such a heterogeneous dataset. This grouping information can be used for domain model recovery as well as model repository management scenarios.

An advantage of our approach is the scalability and tool support. The algorithm complexities range from linear (e.g. VSM construction) to polynomial (hierarchical clustering) with respect to the size of the dataset and of the metamodels in it. Indeed this technique, and more advanced versions thereof, have already been in widespread use in IR for document retrieval and clustering of large collections of data. Moreover, R provides a plethora of efficient and flexible statistical libraries for analysis. (Meta-)metamodel-based construction of the unigram vocabulary and tokenization provides a good amount of reduction in vector space, improving over basic IR indexing. Finally we would like to repeat and emphasize that, although we used the term 'metamodel' clustering throughout the paper (because of the datasets we chose), we regard the metamodels as instances of the Ecore meta-metamodel, thus simply as models. Thus we deal with the generic problem of model comparison and clustering.

### 5.1 Threats to Validity

There are several threats to validity for this study. First of all, the NLP techniques employed might not be accurate enough and need to be improved with features such as context-sensitivity and a domain-specific thesaurus. The fact that we regard metamodel identifiers as bag of words and unigrams, thus ignoring structural relations such as containment and inheritance and semantics, could reduce the accuracy and applicability of our approach in some scenarios. Ignoring the multiplicities and modifiers (e.g. abstract) of model elements also might

lead to a similar shortcoming. Furthermore, the datasets we used are assembled by us; actual datasets that are used in domain model recovery or SPL extraction should be investigated to compare the results. The visualization and manual inspection approach could limit our approach (as it is now) for larger datasets (e.g. $> 1000$ items) and further reduction and visualization techniques might be needed. Last but not the least, the quantitative comparison of the accuracy of different combinations of parameters/components in virtually every step of our approach, and automation of this process would relieve the user from the effort of trial-and-error exploration of the parameters.

## 6 Related Work

Only a few model comparison techniques consider the multiplicity of input models without doing pairwise comparisons, such as N-way merging based on weighted set packing [23]. Feature model extraction [24] and concept mining [1] use NLP to cluster features/concepts. Another technique proposes building domain ontologies as the intersection of graphs of APIs [22], but does not focus on the statistical dimension of problem. Metamodel recovery [13] is another approach which assumes a once existing (but somehow lost) metamodel, and does not hold for our scenario. A technique similar to ours is applied specifically for business process models using process footprints [10], and thus lacks the genericness of our approach. Note that a thorough literature study beyond the technological space of MDE, for instance regarding data schema matching and ontology matching/alignment, is out of scope for this paper and is therefore omitted.

Clustering is considered in the software engineering community mostly within a single body of code [17] or model [26]. A related technique uses clustering for the visualization of Simulink model clones according to the percentage differences and patterns among clones [2]. A very recent approach, which we encountered after publishing our early work, is presented by Basciani et al. [7]. They share most of our objectives, though focusing on repository management. Moreover they use cosine distance of term vectors representing models and HAC for visualization of metamodel repositories. However, they do not report in detail the NLP techniques (e.g. synonym checking) or IR techniques (e.g. weighting) they use. It is left as future work to compare their approach with ours.

## 7 Conclusion and Future Work

In this paper, we have presented a new perspective on the N-way comparison and analysis of models as a first step in domain model recovery. We have proposed a generic approach using the IR techniques VSM and tf-idf enhanced with NLP techniques to uniformly represent multiple metamodels, and apply hierarchical clustering for comparative analysis and visualization of a large dataset. We demonstrated our approach on two real datasets; one of top search results from GitHub and another from the AtlanMod Metamodel Zoo. The results, both

qualitatively for both case studies and quantitatively for the second case study, indicate that our generic and scalable approach is a promising first step for analysing large datasets of models or metamodels.

As future work, we definitely wish to address the points listed as threats to validity. Most notably, the efficiency of different parameters and components of our approach such as various weighting and idf schemes, distance measures and clustering algorithms can be quantitatively evaluated and compared. Another crucial improvement is to incorporate into the analysis both structure and context information (either as n-grams, or tree/graphs) as well as semantics of the metamodel elements. Furthermore, one could investigate the application of our approach for different formalisms such as UML models, and different problems such as model versioning, model merging and model clone or pattern detection.

## References


1. Abebe, S.L., Tonella, P.: Natural language parsing of program element names for concept extraction. In: 2010 IEEE 18th International Conference on Program Comprehension (ICPC), pp. 156–159. IEEE (2010)
2. Alalfi, M.H., Cordy, J.R., Dean, T.R.: Analysis and clustering of model clones: an automotive industrial experience. In: 2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineeringand Reverse Engineering (CSMR-WCRE), pp. 375–378. IEEE (2014)
3. Altmanninger, K., Seidl, M., Wimmer, M.: A survey on model versioning approaches. Int. J. Web Inf. Syst. **5**(3), 271–304 (2009)
4. Babur, Ö., Cleophas, L., Verhoeff, T., van den Brand, M.: Towards statistical comparison and analysis of models. In: Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development, pp. 361–367 (2016)
5. Babur, Ö., Smilauer, V., Verhoeff, T., van den Brand, M.: Multiphysics and multiscale software frameworks: an annotated bibliography. Technical report 15-01, Dept. of Mathematics and Computer Science, Technische Universiteit Eindhoven, Eindhoven (2015)
6. Babur, Ö., Smilauer, V., Verhoeff, T., van den Brand, M.: A survey of open source multiphysics frameworks in engineering. Procedia Comput. Sci. **51**, 1088–1097 (2015)
7. Basciani, F., Di Rocco, J., Di Ruscio, D., Iovino, L., Pierantonio, A.: Automated clustering of metamodel repositories. In: Nurcan, S., Soffer, P., Bajec, M., Eder, J. (eds.) CAiSE 2016. LNCS, vol. 9694, pp. 342–358. Springer, Heidelberg (2016). doi:10.1007/978-3-319-39696-5_21
8. Brunet, G., Chechik, M., Easterbrook, S., Nejati, S., Niu, N., Sabetzadeh, M.: A manifesto for model merging. In: Proceedings of the 2006 International Workshop on Global Integrated Model Management, pp. 5–12. ACM (2006)
9. Deissenboeck, F., Hummel, B., Juergens, E., Pfaehler, M., Schaetz, B.: Model clone detection in practice. In: Proceedings of the 4th International Workshop on Software Clones, pp. 57–64. ACM (2010)
10. Dijkman, R., Dumas, M., van Dongen, B., Käärik, R., Mendling, J.: Similarity of business process models: metrics and evaluation. Inf. Syst. **36**(2), 498–516 (2011)

11. Holthusen, S., Wille, D., Legat, C., Beddig, S., Schaefer, I., Vogel-Heuser, B.: Family model mining for function block diagrams in automation software. In: Proceedings of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools, vol. 2, pp. 36–43. ACM (2014)
12. Jain, A.K., Dubes, R.C.: Algorithms for Clustering Data. Prentice-Hall Inc., Englewood Cliffs (1988)
13. Javed, F., Mernik, M., Gray, J., Bryant, B.R.: Mars: a metamodel recovery system using grammar inference. Inf. Softw. Tech. **50**(9), 948–968 (2008)
14. Klint, P., Landman, D., Vinju, J.: Exploring the limits of domain model recovery. In: 2013 29th IEEE International Conference on Software Maintenance (ICSM), pp. 120–129. IEEE (2013)
15. Kolovos, D.S., Rose, L.M., Matragkas, N., Paige, R.F., Guerra, E., Cuadrado, J.S., De Lara, J., Ráth, I., Varró, D., Tisi, M., Cabot, J.: A research roadmap towards achieving scalability in model driven engineering. In: Proceedings of the Workshop on Scalability in Model Driven Engineering, BigMDE 2013, pp. 2:1–2:10. ACM, New York (2013). http://doi.acm.org/10.1145/2487766.2487768
16. Kolovos, D.S., Ruscio, D.D., Pierantonio, A., Paige, R.F.: Different models for model matching: an analysis of approaches to support model differencing. In: ICSE Workshop on Comparison and Versioning of Software Models, 2009. pp. 1–6. IEEE (2009)
17. Kuhn, A., Ducasse, S., Gírba, T.: Semantic clustering: identifying topics in source code. Inf. Softw. Technol. **49**(3), 230–243 (2007)
18. Lucrédio, D., de M. Fortes, R.P.: Moogle: a metamodel-based model search engine. Softw. Syst. Model. **11**(2), 183–208 (2012)
19. Manning, C.D., Raghavan, P., Schütze, H., et al.: Introduction to Information Retrieval, vol. 1. Cambridge University Press, Cambridge (2008)
20. R Core Team: R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria (2014). http://www.R-project.org/
21. Ramey, J.A.: clusteval: Evaluation of Clustering Algorithms (2012). http://CRAN.R-project.org/package=clusteval, r package version 0.1
22. Ratiu, D., Feilkas, M., Jürjens, J.: Extracting domain ontologies from domain specific apis. In: 12th European Conference on Software Maintenance and Reengineering, 2008, CSMR 2008, pp. 203–212. IEEE (2008)
23. Rubin, J., Chechik, M.: N-way model merging. In: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, pp. 301–311. ACM (2013)
24. She, S., Lotufo, R., Berger, T., Wąsowski, A., Czarnecki, K.: Reverse engineering feature models. In: 2011 33rd International Conference on Software Engineering (ICSE), pp. 461–470. IEEE (2011)
25. Stephan, M., Cordy, J.R.: A survey of model comparison approaches and applications. In: Modelsward, pp. 265–277 (2013)
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, p. 7. ACM (2013)
27. Wild, F.: LSA: Latent Semantic Analysis (2015). http://CRAN.R-project.org/package=lsa, r package version 0.73.1