Model analytics for feature models: case studies for S.P.L.O.T. repository

Önder Babur, Loek Cleophas and Mark van den Brand
Eindhoven University of Technology
Eindhoven, The Netherlands
O.Babur@tue.nl,L.G.W.A.Cleophas@tue.nl,M.G.J.v.d.Brand@tue.nl

ABSTRACT

Model-Driven Engineering and Software Product Lines promote the use of models as central artifacts for a variety of activities including domain analysis and generative software development. As these paradigms gain popularity, the number and variety of models in use increase. Several initiatives to gather models in repositories exist, such as ATL Zoo for metamodels or S.P.L.O.T. for feature models, aiming for public access and reuse. However, as those repositories are only partly or not at all curated, the growing number of models leads to problems such as duplicates a.k.a. clones, and lack of repository overview. This makes both repository management and model searching/reuse very hard. We address this issue for S.P.L.O.T. by adapting SAMOS, our generic model analytics framework for feature model comparison. We perform two exploratory case studies. First, we aim for getting a high level repository overview with large clusters and their domains. Secondly, we try to get clusters of highly similar models, to be interpreted as duplicates or clones. We conclude our approach is applicable for feature models and can improve the use and maintenance of S.P.L.O.T.

KEYWORDS

Model-driven engineering, software product lines, feature models, model comparison, vector space model, clustering, model analytics, model management.

1 INTRODUCTION

Model-Driven Engineering (MDE) and Software Product Lines (SPLs) are paradigms heavily using models for a variety of activities ranging from domain analysis to software development, deployment and testing. While one of the key objectives of such paradigms is the management and reuse of increasingly complex software artifacts, the same problem emerges as they gain popularity and wider adoption: there are more, larger and more complex models in use [8]. Recently, there has been some effort to collect various models in model repositories to facilitate public access and reuse. Notable examples are the ATL Ecore Metamodel Zoo[1], and Software Product Lines Online Tools (S.P.L.O.T.) feature model repository[2] [20]. One problem of such repositories is when they are either partly or not at all curated.

This is particularly evident in S.P.L.O.T.: a quick inspection of the individual models reveals that (1) models usually lack proper metadata on their domains, versions, etc.; (2) there are quite many duplicates/clones/versions of models with no explicit relationship noted. Moreover the number of models in the repository increases rapidly, scaling up the aforementioned problems. These have serious implications in scenarios involving both repository management and use. First of all, there is a lack of repository overview, e.g. what groups of models there are, and to which domains these belong. This type of information would enable repository exploration, facilitating model search and reuse. Secondly, as new models are added, either the model manager or the users themselves are burdened with the manual labeling of the models e.g. with respect to their domains. And lastly, there is a considerable amount of duplicate models, clones arbitrarily copy-pasted, and also various versions of the same models lying around in the repository.

These issues have been raised in the domain of MDE [7, 9]. A promising solution is the automatic comparison of models [24] for gaining some information on the repository dataset such as grouping/subgrouping of models, proximities among models (and groups as well) and outliers. Doing this on a large scale for hundreds of models requires techniques beyond the complex and expensive pairwise comparison such as in [19]; rather it requires approximate but fast and scalable techniques. These include e.g. fragmentation of models into smaller chunks, typically via Information Retrieval(IR)-based and statistical methods such as clustering [7, 9], especially for clone detection [5].

There has been a considerable amount of work in the SPL community on feature model analysis, comparison and use of IR-based techniques, however with several important distinctions. First of all, inspecting the thorough literature study of Benavides et al. [11] on automated analysis of feature models reveals that analysis is mostly performed on a single feature model and some configuration of that, for instance to find out the dead features or valid products. Other approaches involve multiple feature models as input, model comparison is generally perceived based on the configuration semantics (as used by She et al. [23] in contrast with ontological semantics): feature models are transformed into logical formulas, and reasoned about their pairwise relationships such as generalization/specialization [25], or exact differences [3, 12]. Another approach uses EMF Compare to calculate pairwise differences between feature models [13]. An interesting take on feature model comparison is presented by Xing [26], who argues that feature models might evolve over time with changes in both the structure and feature names/descriptions, and applies their generic model differencing technique to feature models using the structural (or ontological according to [23]) information in the models. On the other hand, many researchers have proposed IR and clustering, not for comparing feature models but requirements, product descriptions, or features themselves (e.g. their names, the text in their description) for reverse engineering feature models [4, 23]. Along a line of work mostly on model synthesis and composition [1, 2],

[1] http://web.emn.fr/x-info/atlanmod/index.php?title=Ecore
[2] http://www.splot-research.org/

Bécan et al. utilize IR and NLP techniques in their interactive model synthesis tool [10]. To our best knowledge, there has been no comparable work in the modelling and SPL domains to cluster large numbers of feature models with our objectives and scalability.

In this paper, we attempt to apply our generic model analytics framework to compare the feature models in the S.P.L.O.T. repository. Our goals are twofold; introducing our approach to the SPL community which we believe can benefit from the proposed techniques, and testing the genericness and extensibility of our approach for new model types and datasets. First, we extend our framework with an extraction scheme for feature models using the S.P.L.O.T. Java API for parsing Simple XML Feature Model (SXFM) files. Using many utilities of the framework, notably Natural Language Processing (NLP) tools, we test our approach on the 1034-model dataset in S.P.L.O.T. We perform two case studies: firstly trying to get relatively large sized clusters and their corresponding domains in the repository; and secondly obtaining clusters of very similar models—to be interpreted as duplicates, clones or versions. We conclude our approach is indeed applicable for feature models and can improve the use and maintenance of S.P.L.O.T.

2 ANALYZING FEATURE MODELS

In this section we start with some preliminaries and move on to detail our approach for analyzing and comparing feature models.

2.1 SXFM Feature Models

There are many notations for feature models, starting with the original one by Kang et al. [14], later extended with cardinalities, additional constraints, attributes and so on [21, 22]. As a starting point for this study we take the SXFM notation supported by the models in S.P.L.O.T. A feature model has a feature tree with different types of features in it (Root, Solitaire), optional/mandatory modifier, feature groups with cardinalities (lower and upper bounds) and grouped features in them, and the parent-child relations. They may also contain additional constraints in Conjunctive Normal Form (CNF) clauses. See Figure 2 for an example SXFM feature model with mandatory, optional and grouped features.

2.2 Underlying Concepts of SAMOS Framework

Information Retrieval [17] deals 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 a Vector Space Model (VSM) with the following major components: (1) a vector representation of occurrence of the vocabulary in a document, named *term frequency*, (2) *zones* (e.g. 'author' or 'title'), (3) weighting schemes such as inverse document frequency (idf), and zone weights, (4) NLP techniques for handling compound terms, detecting synonyms and semantically related words.

The VSM allows transforming each document into an n-dimensional vector, resulting in an $m \times n$ matrix for m documents. Over the VSM, document similarity can be defined as the distance (e.g. Euclidean or cosine) between vectors. These can be used for identifying similar groups of documents in the vector space. This unsupervised machine learning (ML) technique is called clustering. Among many clustering methods [17], there is a distinction between flat clustering, where a flat cluster labeling is done, and hierarchical clustering, where a hierarchy of proximities is produced.

Finally, n-grams [18] are used in computational linguistics to build probabilistic models of natural language text, e.g. for estimating the next word given a sequence of words, or comparing text collections based on their n-gram profiles. In essence, n-grams represent a linear encoding of structural context.

2.3 SAMOS Framework for Feature Models

Our generic model analytics framework SAMOS (Statistical Analysis of MOdelS) [5–7] applies the above ideas for models. We have so far used SAMOS for Ecore metamodels, UML class diagrams, state charts and industrial domain specific modelling languages; in the scenarios of domain clustering, data preprocessing and filtering, and notably clone detection [5]. The workflow, as depicted in Figure 1, starts with the extraction of IR-features (note the distinction with features as in feature models) and constraints from a set of input feature models, with a traversal of those models using the SXFM Java Parser Library provided by S.P.L.O.T. We use various schemes and NLP steps such as tokenization, filtering and synonym checking, to populate a VSM after NLP. As a result, each feature model is represented in the VSM as a point in a high dimensional space and model similarity is reduced to a distance calculation. Clustering is applied over these distances. The framework allows configuring several matching schemes (e.g. whether to ignore types, check synonyms) and weighting schemes (e.g. idf or type weights).

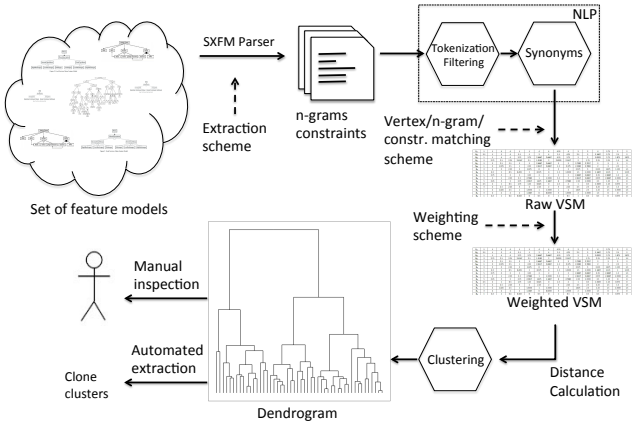

Figure 1: Overview of SAMOS workflow.

2.4 Extracting n-grams and Constraints

We want to extract the information from the features (names, types, cardinalities) and their relation to other features in the hierarchy (i.e. structural context) in the form of n-grams as supported by our framework. Additionally, we want to represent constraints for accurate comparison. Using the SXFM Java parser library of S.P.L.O.T., it is rather straightforward to traverse the feature tree

and generate the information to be used for clustering. We have used the following scheme to extract n-grams and constraints:

- Start with the root node of the tree.
- For each feature traversed, extract its name and type (*Mandatory,Grouped),etc.* as an item of the n-gram chains.
- For each regular child, advance the traversal and add them to the n-gram chains with the cardinality (0..1 for optional, 1..1 for mandatory) captured in the edge information.
- For each grouped feature, advance the traversal and add them to the n-gram chains with group cardinality (0..1 for optional, 1..1 for mandatory). Note that this is an inaccurate simplification due to our choice of using n-grams (due to the current implementation of SAMOS), please see Section 4 on future work to overcome this limitation with more complex features.
- for each clause in the CNF constraints, replace the unique feature ids by the feature names and extract as an unordered set.

We give an example extraction for a model in S.P.L.O.T. (Figure 2) for $n = 1, 2$ and constraints in Table 1. We use a mobile phone feature model, with mandatory features (*Calls*), optional ones (*GPS*), and an alternative feature group (meaning only one should be chosen) with *Basic*, *Colour* or *High Resolution* screen.

Figure 2: A feature model example (constraints not shown).

Table 1: IR-feature extraction: some examples for Figure 2. '~' denotes negation.

type	IR-features
unigram	(Root-Mobile phone) (Mandatory-Calls) (Optional-GPS) (Mandatory-Screen) (Grouped-Basic) …
bigram	(Root-Mobile phone)-(child[1..1])-(Solitaire-Calls) (Root-Mobile phone)-(child[0..1]-(Solitaire-GPS) (Solitaire-Screen)-(child[1..1])-(Grouped-Basic) (Solitaire-Screen)-(child[1..1])-(Grouped-Colour) …
constr	(~GPS,~Basic) (High resolution,~Basic) …

2.5 Rest of the Workflow

Once we obtain the IR-features, the rest of the framework can be used as is for the n-grams. For the constraint sets, we apply the Hungarian algorithm [16] to obtain a best (partial) match score among the sets based on their feature names and negations (each using vertex similarity in SAMOS). In terms of vertex/node matching, i.e. how to compare unigrams with each other, users can choose to check for synonyms via tokenization, filtering, stemming/lemmatization, Levenshtein distance and WordNet[3]; whether types should be exactly the same or ignored altogether. Finally, users can choose to apply type-based weighting (e.g. some parts of the model might be more important such as classes vs. parameters in UML) and idf.

Having set all the above schemes, the framework computes the VSM based on the n-grams extracted. Using this matrix and picking a distance measure (e.g. cosine for domain clustering), clustering is performed in R. Further options are what type of clustering to do (flat vs. hierarchical) and algorithm-specific parameters. The main output of hierarchical clustering is the dendrogram, which can be manually inspected, or *cut* with certain parameters to automatically infer clusters (e.g. for clone detection with threshold values).

3 CASE STUDIES

We performed two exploratory case studies to demonstrate the applicability of our approach for feature models, on the 1034-model dataset in S.P.L.O.T. (as of July 18, 2018[4]).

3.1 Case Study 1 - Repository Overview and Major Domains

In this case study, we want to obtain large groups of related feature models, to be able to identify roughly the domains in the repository (e.g. mobile phone models). Observing that in our case the domain knowledge is captured mostly in the feature names, unigrams ($n = 1$) are adequate here. We have adopted a similar parameter set previously used for clustering the ATL Ecore metamodels [7]: unigrams of names only (no types), NLP including pre-tokenization for compound words, Levenshtein distance for typos, stemming, lemmatization and WordNet for semantic relatedness; normalized log idf weighting, cosine distance and finally hierarchical clustering with average linkage. The procedure for this case study is as follows: (1) cluster the whole dataset with the above settings, (2) perform a filtering pass to cut off the models that are less similar (≥ 0.8 cosine distance, arbitrarily chosen as *high enough* similarity) to the rest of the dataset, and focus on relatively large clusters (≥ 20 models), and (3) perform a second clustering step on the subset and visualize the dendrogram.

Note that the filtering step with specific thresholds for similarities and cluster sizes is necessary, as we have to manually inspect and evaluate the results; manually handling a 1034-item dendrogram with a non-trivial coarse structure within the scope of this work is not feasible. Figure 4 is useful to see the diversity of the models in S.P.L.O.T: there is not much thick branching, for instance dividing the dataset into few large clusters.

[3] https://wordnet.princeton.edu/
[4] snapshot available at http://www.win.tue.nl/~obabur/data/AMMORE18.zip

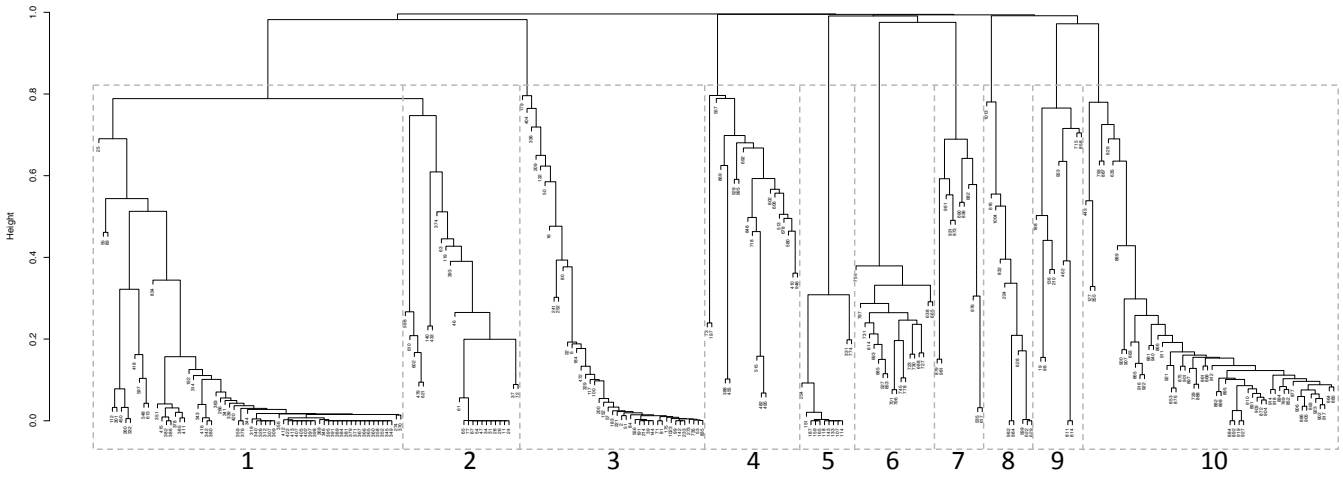

Figure 3: The dendrogram depicting 10 domain clusters.

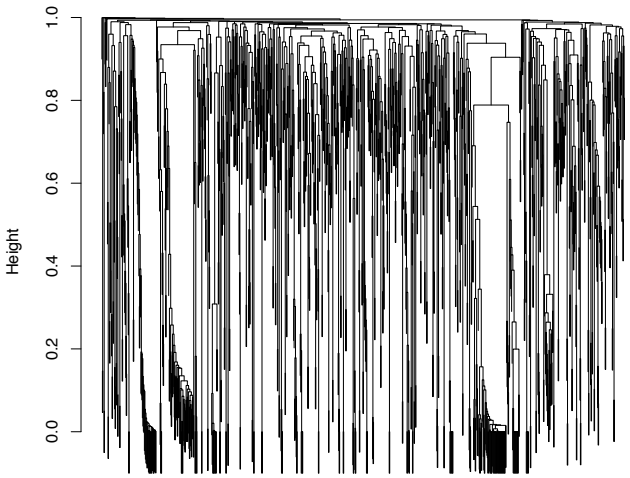

Figure 4: The dendrogram of the 1034 models. Leaves are hung from the joints (which denote the actual similarity) for better visualization; hence some leaves extend below 0.

A Brief Qualitative Evaluation. The filtering steps reduced the dataset size to 275. The resulting dendrogram for clustering those 275 models is given in Figure 3. The interpretation of the dendrogram is that (1) the numbers on the dendrogram correspond to the table row indices of the feature models as given in S.P.L.O.T. table and (2) the joining height of individual branches are the normalized distance (can be considered percentage dissimilarity) between those two individual models or groups of models. Cutting the dendrogram horizontally at height 0.8, we obtain 10 major clusters, as shown in Figure 3 in dashed lined boxes with cluster labels at the bottom. Inspecting the models, we can roughly attribute the following domain labels to the clusters: cluster 1 of mobile media and cluster 2 of mobile phone models, cluster 3 of models with many feature names *Feature-1, Feature-2, . . . Feature-N*, cluster 4 of voting/e-voting models in Portuguese (urna), cluster 5 of models with many feature names *F1, F2, F_1*, and so on; cluster 6 of models in Spanish about real estate (inmobiliaria), cluster 7 of models for marketplaces (mostly in Spanish), cluster 8 of models with a lot of abbreviations and numbers as feature names (see discussion below), cluster 9 of e-shop and e-commerce models, cluster 10 of computer models (mostly in Spanish).

A precise account of the accuracy of this categorization is difficult to give, as the dataset itself is not labeled. Instead we comment about the clusters and some false positives we found by manual inspection. Clusters 1 and 2 are very accurate, and we could find cases where our tool successfully detected typos and NLP-related changes. Cluster 3 has the upper part of the branch (say, higher than 0.5 distance) seemingly less and less relevant; models with few percentages of features with names *Feature-X* are detected as partly relevant to this cluster.

Cluster 4 is also a quite accurate account including non-English (in this case Portuguese) models, although our tool cannot specifically address them at the moment (e.g. with synonym detection in other languages). Cluster 5 ($F's$) is conceptually similar to 3, but does not have as many irrelevant models and is mostly accurate. Clusters 6 and 7 are again mostly accurate except for an outlier numbered 682 (about mass transport). Cluster 8 is the most irrelevant one, where a lot of different models with abbreviations and numbers as feature names are grouped together. Clusters 9 and 10 also seem quite accurate. In the latter, we even identified two models about computers in different languages being correctly clustered together thanks to some shared terms.

A detailed account of the recall for this case study is left as future work. One can arguably increase the recall (at the cost of precision) by relaxing the parameters/thresholds more generously. Note that there are certainly some more domains in the dataset to be discovered, e.g. car and bike feature models. If there are too many domains, it might be practical to handle all of the dataset manually; a semi-automatic way could solve the problem and is left as future work.

3.2 Case Study 2 - Detecting Duplicates and Clones

In this case study, we set the objective to obtain groups of very similar feature models, both content- and structure-wise. We would like to detect duplicates, clones, variants and versions in S.P.L.O.T. , easily seen with a brief inspection of the models in the repository. Exact categorization of the found models into one of these is beyond the scope of this work; we refer to all of those simply as clones. As we want to capture as much information as possible, we turned to use full bigrams (with types, cardinalities) and constraints here. We used (1) no idf weighting, (2) relaxed type matching (with non-exact type matches getting a reduced similarity multiplier). We further used masked Bray-Curtis Distance with a density-based clustering technique (please refer to [5] for this technique and [15] for clone detection in general—omitted due to space constraints). In summary, we ran SAMOS with the clone detection setting on the 1034-model dataset for detecting Type A, B and C clones [5] with respective distance thresholds of 0 (identical except cosmetic changes), 0.1 (slightly different) and 0.3 (somewhat different).

Table 2: Clone detection statistics.

clone type	#clusters	#pairs	#models involved
A	22	64	59
B	60	1472	208
C	90	3320	382

Table 3: Some of the clone clusters.

type	model indices in a cluster
A	811 814
A	763 769 773
A	165 187
A	567 568 571 573 589
A	11 24 26 31 34 54
A	…
B	1029 1030
B	967 970
B	884 892 919 927
B	599 622 629
B	479 602
B	…
C	1032 1034
C	783 794
C	556 735 835
C	11 24 26 28 31 34 37 47 54 61 65 67 72 77 241 374
C	19 50 98 125
C	…

A Brief Qualitative Evaluation. We found a considerable number of clones; we report in Table 2 the number of clone clusters, the total number of clone pairs and models involved in those.[5] This already indicates that a relatively high percentage of the models in S.P.L.O.T. is highly similar to other models in the repository. The actual clone clusters with some examples are given in Table 3. Inspecting a (random sample of) the clone clusters, we were able to trace the following:

- Type A clones: SAMOS was able to detect Type A clones very accurately; we found no incorrect labeling in the validation subset. Manually inspecting the clusters, we found the following changes which led to a Type A classification (implying no significant change): change in the date of creation of the model, feature model name, metadata, constraint names (i.e. not the content), order of elements in the feature tree and the CNF formulas, consistently changed feature id's (which lead to e.g. completely different looking constraint formulas), and cosmetic changes in feature names (e.g. upper/lower and snake/camel casing).
- Type B clones: SAMOS detected clones with a variety of changes, ranging from simple modification of cardinalities, textual changes in feature names (e.g. typos, additional tokens) to addition or removal of features and constraints, and moving of features to elsewhere in the feature tree as well. Although all the clones we inspected were relevant, some might arguably be categorized as higher level, namely type C (see discussion about weighting and feature groups in Section 4).
- Type C clones: SAMOS is again generally accurate in finding higher percentage of addition, removal, or changes in feature trees and constraints, although we identify certain shortcomings. SAMOS (1) treats feature names such as *Feature-1* and *Feature-2* as highly similar, which leads to an inaccurate Type C classification (see e.g. line 3 of type C clones in Table 3). Also due to the simplification of grouped features in the form of bigrams, SAMOS is not able to distinguish very well between e.g. grouped features in a strict alternative feature group of cardinality 1..1, and the same features moved outside as mandatory features (again with cardinalities 1..1). The problem with weighting as mentioned above might lead to some misjudgment and is subject to improvement.

4 DISCUSSION AND FUTURE WORK

The case studies show our approach can provide an insight into the feature models in S.P.L.O.T., in terms of repository overview and domain decomposition, and of duplicates a.k.a. clones. We extract the information captured in the feature names, the ontological hierarchy of the feature tree and the constraints (syntax only); and use this to efficiently calculate approximate similarities among models. Here we provide no quantitative evaluation on the accuracy of our approach partly due to the lack of a labeled dataset and the exploratory nature of the study. A brief qualitative evaluation yet reveals our approach is indeed effective to a considerable extent with room for improvements. It would be interesting to quantitatively evaluate the effect of different settings, given a labeled dataset (e.g. feature models with explicit domains for the first case study, or mutated feature models for clone detection). In this section we discuss several limitations of and improvements for our approach.

[5] See http://www.win.tue.nl/~obabur/data/AMMORE18.zip for the full list of clones

Grouped features, configuration semantics: Given our choice of bigram representation, we inaccurately extract group cardinalities for each grouped feature. An improvement would be to switch to tree representations in SAMOS (still in development) to properly capture those. Furthermore, we do not perform any inference and compare the syntactic constructs as is. Hence, it should be further investigated how we can incorporate the inferred information, though then the approach would would move towards comparing knowledge bases. Another further step would be incorporating attributes in extended feature models for comparison, which is not supported by the S.P.L.O.T. dataset, and left as an open problem.

Weighting, fine tuning: At the moment we did not use any weighting scheme in this paper, such as type-based weighting (e.g. constraints having less weight than the feature tree) as supported by SAMOS, but also advanced ones. An initial idea for the latter would be depth-dependent weighting, i.e. features lower in the tree hierarchy get lower weights, hence attributing more importance to higher level features (which are arguably more general or abstract, e.g. as mere structural units, more coarse grained/architecture-related). Inspecting the results of the clone detection, we believe a fine-tuned weighting scheme could improve the clone classification, especially around boundaries between Type B, C and higher thresholds.

NLP settings: The S.P.L.O.T. dataset brings several new challenges for our framework's NLP capabilities, notably due to its multi-language heterogeneous nature. There are models in English, Spanish, Portuguese, Indonesian, etc. in the repository, which renders our English-based NLP tools inadequate. In an orthogonal direction for improvement, the framework could be extended with multi-language NLP, including e.g. tokenizers and even cross-language synonym checkers. The features labeled as *Feature-1* or *F1*, or cryptically abbreviated, pose yet another challenge.

Threats to validity. There are several threats to validity for our work, mostly stemming from the exploratory approach. The settings we have chosen for the case studies may not be the most efficient and accurate ones, but were chosen mostly for simplicity and demonstration purposes. A quantitative evaluation of different parameters and thresholds, and more importantly on labelled datasets (e.g. explicit domain labels for case study 1, clones for case study 2) would be required for a more precise account.

5 CONCLUSION

In this paper we have presented an application of our generic model clustering technique to feature models. We have extended our framework to extract information from feature models and efficiently but approximately compare models. With two exploratory case studies on the 1034-model dataset in the S.P.L.O.T. repository, we get (1) a repository overview and major domains therein, (2) very similar models in the repository such as duplicates and clones. Based on the studies, we conclude that our approach is applicable for clustering feature models to a great extent. Following the two objectives we set in the beginning, we both confirm the genericity and applicability of our approach for different types of models, and provide a new perspective on the comparison of feature models for the SPL community. Indeed, our approach can help with the use and maintenance of emerging repositories such as S.P.L.O.T.

REFERENCES

[1] Mathieu Acher, Benoit Baudry, Patrick Heymans, Anthony Cleve, and Jean-Luc Hainaut. 2013. Support for reverse engineering and maintaining feature models. In *Int. Workshop on Variability modelling of Software-intensive Systems*. ACM, 20.

[2] Mathieu Acher, Benoit Combemale, Philippe Collet, Olivier Barais, Philippe Lahire, and Robert B France. 2013. Composing your compositions of variability models. In *Int. Conf. on Model Driven Engineering Languages and Systems*. Springer, 352–369.

[3] Mathieu Acher, Patrick Heymans, Philippe Collet, Clément Quinton, Philippe Lahire, and Philippe Merle. 2012. Feature model differences. In *Int. Conf. on Advanced Information Systems Engineering*. Springer, 629–645.

[4] Vander Alves, Christa Schwanninger, Luciano Barbosa, Awais Rashid, Peter Sawyer, Paul Rayson, Christoph Pohl, and Andreas Rummler. 2008. An exploratory study of information retrieval techniques in domain analysis. In *Software Product Line Conference, 2008. SPLC'08. 12th International*. IEEE, 67–76.

[5] Önder Babur. 2018. Clone Detection for Ecore Metamodels using N-grams. In *Proc. of the 6th Int. Conf. on Model-Driven Engineering and Software Development, 2018*. 411–419.

[6] Önder Babur and Loek Cleophas. 2017. Using n-grams for the Automated Clustering of Structural Models. In *43rd Int. Conf. on Current Trends in Theory and Practice of Computer Science*. 510–524.

[7] Önder Babur, Loek Cleophas, and Mark van den Brand. 2016. Hierarchical Clustering of Metamodels for Comparative Analysis and Visualization. In *Proc. of the 12th European Conf. on Modelling Foundations and Applications, 2016*. 2–18.

[8] Önder Babur, Loek Cleophas, Mark van den Brand, Bedir Tekinerdogan, and Mehmet Aksit. 2018. Models, More Models, and Then a Lot More. In *Software Technologies: Applications and Foundations*, Martina Seidl and Steffen Zschaler (Eds.). Springer International Publishing, Cham, 129–135.

[9] Francesco Basciani, Juri Di Rocco, Davide Di Ruscio, Ludovico Iovino, and Alfonso Pierantonio. 2016. Automated clustering of metamodel repositories. In *Int. Conf. on Advanced Information Systems Engineering*. Springer, 342–358.

[10] Guillaume Bécan, Sana Ben Nasr, Mathieu Acher, and Benoit Baudry. 2014. WebFML: synthesizing feature models everywhere. In *Proceedings of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools-Volume 2*. ACM, 112–116.

[11] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. 2010. Automated analysis of feature models 20 years later: A literature review. *Information Systems* 35, 6 (2010), 615–636.

[12] Johannes Bürdek, Timo Kehrer, Malte Lochau, Dennis Reuling, Udo Kelter, and Andy Schürr. 2015. Reasoning about product-line evolution using complex feature model differences. *Automated Software Engineering* (2015), 1–47.

[13] Nicolas Dintzner, Arie van Deursen, and Martin Pinzger. 2015. Analysing the Linux kernel feature model changes using FMDiff. *Software & Systems Modeling* (2015), 1–22.

[14] Kyo C Kang, Sholom G Cohen, James A Hess, William E Novak, and A Spencer Peterson. 1990. *Feature-oriented domain analysis (FODA) feasibility study*. Technical Report. DTIC Document.

[15] Rainer Koschke. 2007. Survey of research on software clones. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[16] Harold W Kuhn. 1955. The Hungarian method for the assignment problem. *Naval research logistics quarterly* 2, 1-2 (1955), 83–97.

[17] Christopher D Manning, Prabhakar Raghavan, Hinrich Schütze, et al. 2008. *Introduction to information retrieval*. Vol. 1. Cambridge university press Cambridge.

[18] Christopher D Manning and Hinrich Schütze. 1999. *Foundations of statistical natural language processing*. Vol. 999. MIT Press.

[19] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. 2002. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Data Engineering, 2002. Proc. 18th Int. Conf. on*. IEEE, 117–128.

[20] Marcilio Mendonca, Moises Branco, and Donald Cowan. 2009. SPLOT: software product lines online tools. In *Proc. of the 24th ACM SIGPLAN Conf. Companion on Object oriented Prog. Systems Languages and Applications*. ACM, 761–762.

[21] Pierre-Yves Schobbens, Patrick Heymans, Jean-Christophe Trigaux, and Yves Bontemps. 2007. Generic semantics of feature diagrams. *Computer Networks* 51, 2 (2007), 456–479.

[22] Christoph Seidl, Tim Winkelmann, and Ina Schaefer. 2016. A software product line of feature modeling notations and cross-tree constraint languages. *Modellierung* (2016).

[23] Steven She, Rafael Lotufo, Thorsten Berger, Andrzej Wasowski, and Krzysztof Czarnecki. 2011. Reverse engineering feature models. In *Software Engineering (ICSE), 2011 33rd International Conference on*. IEEE, 461–470.

[24] Matthew Stephan and James R Cordy. 2013. A Survey of Model Comparison Approaches and Applications.. In *Modelsward*. 265–277.

[25] Thomas Thüm, Don Batory, and Christian Kästner. 2009. Reasoning about edits to feature models. In *31st Int. Conf. on Software Engineering*. IEEE, 254–264.

[26] Zhenchang Xing. 2010. Model comparison with GenericDiff. In *Proc. of the IEEE/ACM Int. Cont. on Automated Software Engineering*. ACM, 135–138.