Model Fragment Reuse Driven by Requirements

Raúl Lapeña[1], Jaime Font[1], Carlos Cetina[1], and Óscar Pastor[2]

[1] SVIT Research Group, Universidad San Jorge
Autovía A-23 Zaragoza-Huesca Km.299
50830 Villanueva de Gállego (Zaragoza), Spain
{rlapena,jfont,ccetina}@usj.es

[2] Centro de Investigación en Métodos de Producción de Software
Universitat Politècnica de València
Camino de Vera, s/n, 46022 Valencia, Spain
opastor@pros.upv.es

Abstract. Clone-and-Own is a common practice in families of software products, where parts from legacy products are reused in new developments. In industrial scenarios, CAO consumes high amounts of time and effort, not guaranteeing good results. We propose a novel approach, Computer Assisted CAO for Models (CACAO4M), that uses a Multi-Objective Evolutionary Algorithm (MOEA) with two objectives (Model Fragment Similitude, and Model Fragment Understandability) to rank relevant model fragments for reuse. We evaluated our approach in the industrial domain of train control software. Our approach outperforms the results of a baseline that uses only the Model Fragment Similitude metric, which encourages us to further research in this direction.

Keywords: Software Reuse, Feature Location, Model Driven Development

1 Introduction

Clone-And-Own (CAO) [1] is a common practice in the development of new products, consisting of adapting elements from legacy products in new product implementations. Reuse enables faster software development and easier tracking of projects, and helps maintain the development style and conventions consistent between products. In practice, CAO is carried out manually, relying on developers' knowledge of the family. In industrial scenarios, engineers tasked with new developments often lack knowledge over the entirety of the family, making CAO consume high amounts of time and effort, without guaranteeing good results.

This paper presents Computer Assisted Clone-And-Own for Models (CACAO4M), a novel approach for software families where products are developed through Model-Driven Development (MDD). The approach leverages the Multi-Objective Evolutionary Algorithm (MOEA) [2] technique to rank relevant model fragments for the requirements of a new development with two objectives: (1) the similitude of model fragments to the provided requirement, and (2) the understandability of model fragments from the perspective of a software engineer.

The results of our approach are evaluated in the domain of train control software with our industrial partner, CAF (`http://www.caf.net/en`), a worldwide provider of railway solutions, and compared against those of a baseline that takes in account only the similitude of model fragments to the provided requirement. The results of our approach improve those of the baseline, providing engineers with model fragments that are applicable to the problem requirement and easier to understand than those of the baseline, encouraging further research.

Through our work, Section 2 presents the background, Section 3 details our approach, Section 4 evaluates our approach, Section 5 gathers the related works, and Section 6 concludes the paper.

2 Background

This section presents the Train Control and Management Language (TCML) that formalizes the products from our industrial partner. It has the expressiveness required to (1) describe the interaction between train equipment, and (2) specify non-functional aspects. We present an equipment-focused simplified subset of TCML, along with a running example.

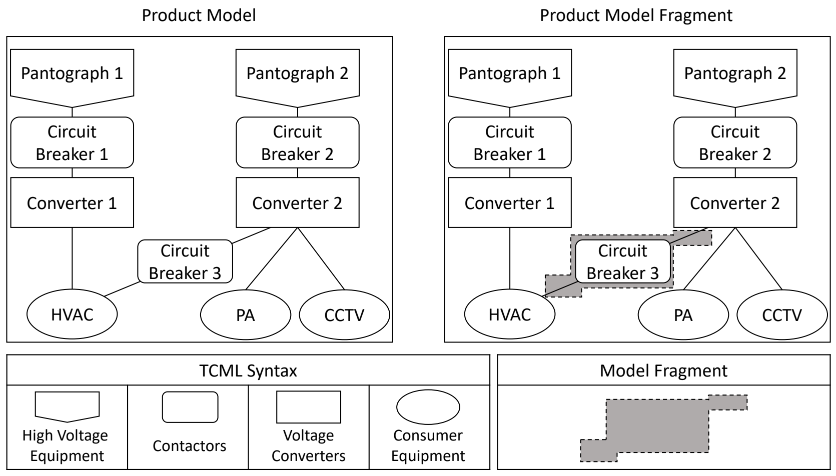


Fig. 1. Example of TCML model and model fragment

Fig. 1 depicts a real-world example model. The right part of Fig. 1 shows an example model fragment that realizes the "converter assistance" requirement, which allows the passing of current from one converter to equipment assigned to its peer to cover overload or failure. To formalize the model fragments used by CACAO4M, we use the Common Variability Language (CVL) [3], which defines variants of a base model by replacing variable parts with alternative model replacements found in a library.

3 Approach

Fig. 2 presents an overview of CACAO4M. As inputs, we use one requirement for a new product in the family, and the models that implement the legacy products in the family. Our approach runs in three steps: (1) **Initialization**, (2) application of **Genetic Operations**, and (3) **Fitness Function** assessing. The last two steps of the approach are repeated until the solution converges to a certain stop condition. When this occurs, the genetic algorithm provides a model fragment list, ranked according to the objectives, for the requirement.

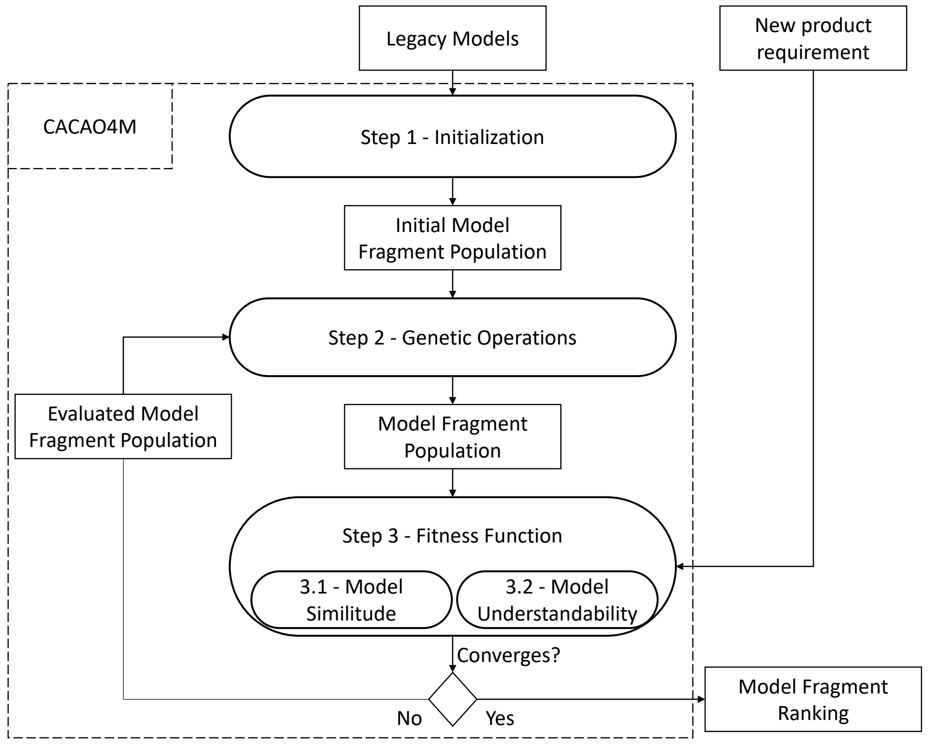

Legacy Models
New product requirement
CACAO4M
Step 1 - Initialization
Initial Model Fragment Population
Step 2 - Genetic Operations
Evaluated Model Fragment Population
Model Fragment Population
Step 3 - Fitness Function
3.1 - Model Similitude
3.2 - Model Understandability
Converges?
No
Yes
Model Fragment Ranking

Fig. 2. Approach Overview

3.1 Initialization & Genetic Operations

The first step of our approach generates an initial collection of model fragments, by randomly extracting parts of the legacy models.

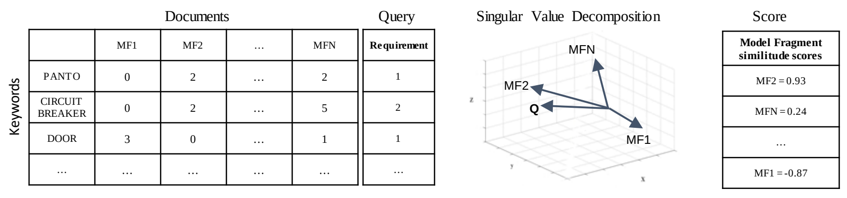
The second step of our approach generates a set of model fragments that could realize the requirement. The generation of new model fragments is done by applying a set of three genetic operators, adapted to work over model fragments: selection of parents, crossover, and mutation. The **selection operator** picks the best candidates from the population as input for the rest of operators. We follow the wheel selection mechanism [4], where each model fragment from the population has a probability of being selected proportional to its fitness score. The **crossover operation** enables the creation of a new individual by combining the genetic material from two parent model fragments. The **mutation operator** is used to imitate the mutations that randomly occur in nature when new individuals are born. The operations are taken from [5] and [6] respectively, where their application to models is detailed.

3.2 Model Fragment Fitness

The third step of the approach assesses each of the candidate model fragments, ranking them according to a fitness function. Our approach presents a fitness function based on two objectives: (1) the degree of similitude of the model fragment to the requirement, and (2) the understandability of the model fragment.

3.2.1 Model Fragment Similitude

To assess the relevance of each model fragment with relation to the provided requirement, we apply Latent Semantic Indexing (LSI) [7]. LSI constructs vector representations of a query and a corpus of text documents by encoding them as a term-by-document co-occurrence matrix. In our approach, terms are keywords extracted from requirements through natural language processing techniques, the documents are generated from the model fragments by extracting the terms that correspond to the elements that conform them, and the query is the provided requirement. Once the matrix is built, it is normalized and decomposed into a set of vectors using a matrix factorization technique called Singular Value Decomposition (SVD) [7]. The similarity degree between the query and each document is calculated through the cosine between the vectors that represent them. Fig. 3 shows an example of co-occurrence matrix, taken from our approach. Fig. 3 also shows the result of applying the SVD technique to the matrix, and the scores associated to each model fragment.



Keywords	MF1	MF2	...	MFN	**Requirement**
PANTO	0	2	...	2	1
CIRCUIT BREAKER	0	2	...	5	2
DOOR	3	0	...	1	1
...	...	...	...	...	...

Model Fragment similitude scores
MF2 = 0.93
MFN = 0.24
...
MF1 = -0.87

Fig. 3. LSI example

3.2.2 Model Fragment Understandability

In order to measure the Understandability of a TCML model fragment, we measure its size by accounting the amount of lines, shapes, and labels that appear in the model fragment. As an example, we highlight the calculations for the model on Fig. 1. To compute the Understandability metric, we take in account the number of lines (9) and the number of shapes (10), for a total of 19 model elements.

4 Evaluation

This section evaluates our approach by applying it to a case study from our industrial partner consisting of 23 trains, each one having an associated requirements specification document, and an associated model. Requirements from the document are implemented through model fragments in the model. The documents specify, on average, around 420 requirements each. The models comprise, on average, around 1200 elements each.

4.1 Experimental Setup

Fig. 4 shows the steps followed to evaluate our approach. First, roles are assigned to products in the product family. One product acts as the new product and the rest act as legacy products. The models of the products that act as legacy products, and one requirement of the product that acts as the new product are used to perform CACAO4M, while the model fragment that implements the latter is kept apart to be used as an oracle. Therefore, in order for a product to act as an oracle, it is necessary to have the mapping between its requirements and the model fragments that implement each of them. From the 23 products in the family, the mapping is available for 4. These products are the ones that can be used as oracles in our family.

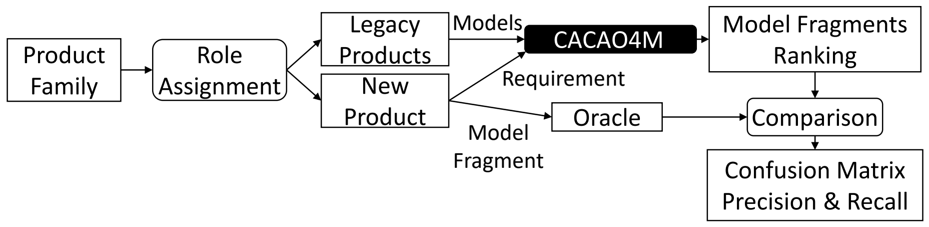


Fig. 4. Evaluation Steps

Then, CACAO4M performs the steps described in our approach to provide a model fragment ranking for the requirement of the new product. We carry out the genetic algorithm inside CACAO4M, weighing the two metrics (Model Similitude and Model Understandability) as 90% - 10%. We also apply CACAO4M with a 100% - 0% weighing, to simulate a Single Objective Evolutionary Algorithm (SOEA) where only Model Similitude is taken in account. The SOEA is considered as the baseline against which the results of the MOEA are compared.

Finally, the first model fragment in the ranking for each result is compared with the oracle model fragment, in order to obtain a confusion matrix. A confusion matrix is a table used to describe the performance of a classification model (in this case, our algorithms) on a set of test data (the resulting model fragments) for which the true values are known (from the oracle). Each solution outputted by the algorithm is a model fragment composed of a subset of the model elements that are part of the product model (where the requirement is being located). Since the granularity is at the level of model elements, each model element presence or absence is considered as a classification.

The confusion matrix distinguishes between the predicted values and the real values classifying them into four categories: (1) **True Positive (TP):** predicted true - real true; (2) **False Positive (FP):** predicted true - real false; (3) **True Negative (TN):** predicted false - real false; and (4) **False Negative (FN):** predicted false - real true. The evaluated performance metrics are:

(1) **Precision**: number of elements from the solution that are correct according to the ground truth, expressed as

$$Precision = \frac{TP}{TP + FP}$$

(2) **Recall**: number of elements of the solution retrieved by the proposed solution, expressed as

$$Recall = \frac{TP}{TP + FN}$$

(3) **F-measure**: harmonic mean of precision and recall, expressed as

$$F - measure = \frac{2 * TP}{2 * TP + FP + FN}$$

We perform our evaluation separately for every requirement in an oracle, and for all the 4 possible oracles individually.

4.2 Results

The LSI 90% & U 10% (MOEA) achieves the best results, providing a mean precision value of 55.34%, a recall value of 49.98%, and a combined F-measure of 52.52%; while the LSI 100% & U 0% (SOEA) achieves a mean precision value of 53.21%, a recall value of 49.73%, and a combined F-measure of 51.41%.

5 Related Work

Feature location approaches in a product family such as the one presented in [8] center their efforts in finding the code that implements a feature between the different products by combining techniques such as FCA and LSI. We are not interested in the code representation of a feature in the family, but in locating the most relevant model fragments that implement a requirement.

Works as [9] focus on the location of features over models by comparing the models with each other to formalize the variability among them in the form of a Software Product Line. We do not locate features, but model fragments that implement requirements, and our goal is not to formalize variability, but to help engineers develop requirements through model fragment Clone-And-Own.

Font et al. [5] use a SOEA to locate features among a family of models in the form of a variation point. Their approach is refined in [6], where a SOEA is used to find sets of suitable feature realizations. The presented approach, in contrast, locates model fragments that are relevant for the development of a single requirement. The presented approach also differs from [5] and [6] both technique and metrics, by using a MOEA, with a fitness function that combines Model Similitude and Model Understandability.

In [10], Lapeña et. al use POS Tagging in combination with an adapted two-step LSI to obtain rankings of methods for all the requirements of a new product in a product family. In the presented work, we obtain only one ranking for one requirement on demand. Plus, this approach uses a MOEA, against the modified LSI in [10]. Finally, the scope of this work is centered around finding models that can be used to implement a particular requirement, not finding relevant code for requirements implementation.

6 Conclusions

Clone-And-Own (CAO) is a common practice in the development of new products in families of software products. In practice, it is carried out manually and relies on human factors, consuming high amounts of time and effort without guaranteeing good results. This paper presents Computer Assisted Clone-And-Own for Models (CACAO4M), a novel approach that leverages a MOEA to rank relevant model fragments for the development of particular requirements for a new product. CACAO4M assesses the similitude of model fragments to the provided requirement, and their understandability. Through our approach, we aim to prioritize the model fragments that are easier to understand from the perspective of a software engineer. Our MOEA approach is evaluated against a SOEA baseline that takes in account only model fragment similitude, outperforming the baseline in every performance indicator. Model Understandability provides a way of locating model fragments that are applicable to the problem requirement and easier to understand than those retrieved by the SOEA. Results encourage us to work further in this direction.

Acknowledgments

This work has been partially supported by the Ministry of Economy and Competitiveness (MINECO) through the Spanish National R+D+i Plan and ERDF funds under the project Model-Driven Variability Extraction for Software Product Line Adoption (TIN2015-64397-R).

References

1. Antkiewicz, M., Ji, W., Berger, T., Czarnecki, K., Schmorleiz, T., Lämmel, R., Stănciulescu, t., Wąsowski, A., Schaefer, I.: Flexible product line engineering with a virtual platform. In: Companion Proceedings of the 36th International Conference on Software Engineering. ICSE Companion 2014, New York, NY, USA, ACM (2014) 532–535
2. Fonseca, C.M., Fleming, P.J., et al.: Genetic algorithms for multiobjective optimization: Formulationdiscussion and generalization. In: ICGA. Volume 93., Citeseer (1993) 416–423
3. Haugen, Ø., Møller-Pedersen, B., Oldevik, J., Olsen, G.K., Svendsen, A.: Adding standardized variability to domain specific languages. In: Software Product Lines, 12th International Conference, SPLC 2008, Limerick, Ireland, September 8-12, 2008, Proceedings. (2008) 139–148
4. Affenzeller, M., Winkler, S.M., Wagner, S., Beham, A.: Genetic Algorithms and Genetic Programming - Modern Concepts and Practical Applications. CRC Press (2009)
5. Font, J., Arcega, L., Haugen, Ø., Cetina, C.: Feature location in model-based software product lines through a genetic algorithm. In: Software Reuse: Bridging with Social-Awareness - 15th International Conference, ICSR 2016, Limassol, Cyprus, June 5-7, 2016, Proceedings. (2016) 39–54
6. Font, J., Arcega, L., Haugen, Ø., Cetina, C.: Feature location in models through a genetic algorithm driven by information retrieval techniques. In: Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems. MODELS '16, New York, NY, USA, ACM (2016) 272–282
7. Landauer, T.K., Foltz, P.W., Laham, D.: An introduction to latent semantic analysis. Discourse processes **25**(2-3) (1998) 259–284
8. Xue, Y., Xing, Z., Jarzabek, S.: Feature location in a collection of product variants. In: 19th Working Conference on Reverse Engineering, WCRE 2012, Kingston, ON, Canada, October 15-18, 2012. (2012) 145–154
9. Wille, D., Holthusen, S., Schulze, S., Schaefer, I.: Interface variability in family model mining. In: 17th International Software Product Line Conference co-located workshops, SPLC 2013 workshops, Tokyo, Japan - August 26 - 30, 2013. (2013) 44–51
10. Lapeña, R., Ballarín, M., Cetina, C.: Towards clone-and-own support: locating relevant methods in legacy products. In: Proceedings of the 20th International Systems and Software Product Line Conference, SPLC 2016, Beijing, China, September 16-23, 2016. (2016) 194–203