Software Clustering by Example

Martin Faunes, Marouane Kessentini, Houari Sahraoui
DIRO, Université de Montréal
Montréal, QC, Canada
{faunescm, kessentm, sahraouh}@iro.umontreal.ca

ABSTRACT

We model software clustering problems in a setting, where elements of a software system form a graph to be partitioned in order to derive high-level abstractions. We extend this formulation in a way that the graph partitioning solutions are evaluated by the degree of their conformance with past clustering cases given as examples. We provide a concrete illustration of this formulation with the problem of object identification in procedural code, for which we obtained better results than a clustering approach.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement —*Restructuring, reverse engineering, and reengineering*

General Terms

Design

Keywords

Software clustering, Example-based software engineering

1. INTRODUCTION

For more than 20 years, researchers have been proposing approaches and algorithms to automatically derive more abstract constructs from existing low-level software elements. In general, deriving such abstract constructs is treated as a clustering problem, where low-level elements are grouped together according to a given objective function. In modern software engineering, abstract elements such as classes and components generally reify actual entities or functions of the application domain. Consequently, their existence obeys the application semantics. If we want to obtain them by clustering existing low-level elements in the code, the clustering objective function should approximate somehow the application semantics from the code. The more a group of basic elements conforms to an application entity according to the approximation method, the more it is considered as an acceptable abstraction.

The most commonly used approximation heuristic in software clustering (SC) problems is that elements that are close structurally are also close semantically, i.e., they form an abstraction that represents an application entity or function. Although the existing approximation methods give good results in some clustering problems, they generally produce a lot of false positives. In this paper, we propose a new formulation of software-clustering problems. In this formulation, we view the clustering as a grouping process guided by the similarity with past clustering examples. We illustrate our proposal with the well-know clustering problem of object identification in procedural code. Our evaluation showed that the majority of identified objects are correct.

2. CLUSTERING BY EXAMPLE

The majority of software clustering contributions is based on the structure-semantics-equivalence hypothesis. In this context, groups of elements that are structurally dependent are viewed as abstractions corresponding to application domain entities. In this setting, the contributions can be summarized as: given a set of elements composing a software and the dependency relationships between them, find groups (clusters) of elements that maximize an objective function and/or satisfy a set of constraints (e.g., [1]). There are many ways to formally represent the problem of software clustering. The most common way is to view it as a graph partitioning problem (e.g., [3, 2]).

The software to be clustered defines a graph $G(V, E)$. The set of vertexes V represents the elements of interest in the software and the set of edges $E \subseteq V \times V$ represent the dependencies between these elements. The graph G is typically a directed graph. For a clustering problem, different types of elements in a software system could be of interest. Consequently, the vertexes in V and edges in E are typed. Moreover, as stated by Mitchell in [4], for some clustering problems, edges in the graph could be weighted to measures the strength of the dependency between two elements. For instance, the weight could indicate the number of method invocations between two classes. The partition of a graph G into m clusters is defined as $P_G = \{K_1, K_2, \cdots, K_m\}$ where each K_i is a cluster corresponding to a sub-graph of G such that each cluster K_i is a non empty sub-graph of G, each vertex in G belongs to one and only one cluster.

An objective function is necessary to evaluate the quality of a partition and to guide the partitioning process. In general, the objective function involves a vector M of metrics to be measured on a clustering solution. Each metric defines an objective to reach. To combine this multiple objectives into a single value, the most common technique is to use a vector W of weights that define the importance of each metric. Cohesion and coupling are the most used metrics for clustering evaluation. In conclusion, following the structural-semantics-equivalence hypothesis, almost all the approaches aims at minimizing or maximizing an objective function defined to measure structural dependencies inside and between clusters starting from a dependency graph. We propose a different approach based on the hypothesis that examples can help recovering part of the semantic proximity. The idea behind our clustering process is to compare configurations of groups of elements to configurations in an example base that led to abstract entities.

We define a base of examples as a set BE of n pairs $s_e = \{G_e, P_{Ge}\}$ of already clustered software represented by a dependency graph G_e and its corresponding partition P_{Ge}, formally, $BE = \{s_e \mid s_e = \{G_e, P_{Ge}\}, 1 \leq e \leq n\}$. The example base is used as set of clusters EK coming from different systems with $EK = \{Q_j \in \bigcup P_{Ge} \mid s_e \in BE\}$. Software elements represented by vertexes in V of $G(V, E)$ are grouped together by similarity with cluster examples $Q_j \in EK$. To this end, we define a function $ave : V \rightarrow KE$ that assigns an example cluster to each element in the graph to partition. The clustering process produces a partition P_G of the graph $G(V, E)$ with the principle that vertexes with the same assigned cluster $Q_j \in KE$ form a cluster $K_i \in P_G$. The objective function f is defined in terms of similarity between the groups of elements of the system to partition with the associated cluster examples. It could be calculated as the weighted average of similarities of these groups. To give equal chances to each group, the similarity is normalized by the size of the groups ($f = \sum_{i=1}^{k} \mid K_i \mid Sim(K_i, Q_j) / \mid V \mid$).

The similarity Sim between a candidate cluster K_i and an example cluster Q_j is defined as a function of the similarities between their respective elements. It is a variation of the graph matching function defined in [21]. Formally,

$$Sim(K_i, Q_j) = \frac{1}{|K_i|} \sum v \in K_i \max_{q \in Q_j} vSim(v, q) \in [0, 1]$$

The function $vSim(v, q)$ compares a vertex $v \in V_i$ of K_i to a vertex $q \in V_j$ of Q_j. $vSim(v, q)$ equals zero if the types of v and q are different, that is, if v and q are not comparable. Otherwise, $vSim(v, q)$ will match the edges $EV(v)$ of v and $EV(q)$ of q and will return the ratio of matched edges over the total edges of v and q.

3. APPLICATION

In this section, we show how this formulation could apply to a specific clustering problem, namely, object identification in procedural code. The intuition behind many solutions to this problem is that if a subset of procedures accesses the same variables, this is an indication that the variables define the state of an object and the procedures its behavior (e.g., [5]). The software to be clustered is defined as a graph $G(V, E)$ where the vertexes are procedures and variables and the edges procedure calls and variable accesses. The base of example BE contains a set of procedural programs $\{G_e, P_{Ge}\}$ represented as graphs and the corresponding partitions. The space of all the possible partitions is very large. To explore this space, a heuristic search is suited. In this illustration we use a hybrid method that combines Particle Swarm Optimization (PSO) and Simulated Annealing (SA). First, we perform a global heuristic search by PSO to reduce the search space and select an initial solution. Thena local heuristic search is done using SA to refine this solution.

To evaluate our example-based clustering, we selected 10 C programs from the website Planet Source Code [1]. Each program was parsed and a corresponding graph was derived. Then, we manually identified object-like structures in each program and defined a partition of its elements accordingly. To measure the correctness of the example-based clustering, we used a 10-fold cross-validation procedure. For each fold, we use 9 programs as the base of examples and identify the objects in the tenth program (a different one in each fold). The objects identified automatically are compared to those found manually. The correctness for each fold is calculated as the proportion of composing elements that are assigned to good objects. The global correctness is derived as the average of the 10 fold's correctness values. Correctness varies between 66% and 100% depending on the programs. The identification in small programs is correctly done (100%). It was good for large programs (around 80%). Programs with less good values are the average ones. 7 over the 10 programs have identification correctness greater than 80%.

We also compared our approach to classical, metric-based ones. For all the 10 programs, the correctness of our approach is significantly better with our new formulation than with the classical one. The difference is even more important for larger programs. This is a clear indication that similarity with previous examples, combined with the structural dependencies, could improve the quality of the clustering approaches in SC problems

4. CONCLUSIONS

In this paper, we first propose a general formulation of the SC problems. Then, we extend this formulation by defining the clustering objective functions in terms of similarity with previous clustering examples rather than only the structural proximity. Our formulation is general and could be used for various SC problems. We illustrate our approach with the well-known problem of object identification.

5. REFERENCES

[1] H. Abdeen, S. Ducasse, H. Sahraoui, and I. Alloui. Automatic package coupling and cycle minimization. In *Proceedings of the 2009 16th Working Conference on Reverse Engineering*, pages 103–112, 2009.

[2] I. Czibula and G. Serban. Improving systems design using a clustering approach. *Int. Journal of Computer Science and Network Security*, 6(12):40–49, 2006.

[3] M. Harman, R. M. Hierons, and M. Proctor. A new representation and crossover operator for search-based optimization of software modularization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1351–1358, 2002.

[4] B. S. Mitchell. *A heuristic search approach to solving the software clustering problem.* PhD thesis, 2002.

[5] H. Sahraoui, H. Lounis, W. Melo, and H. Mili. A concept formation based approach to object identification in procedural code. *Automated Software Engg.*, 6:387–410, 1999.

[1]www.planet-source-code.com