Placement of Entities in Object-oriented Systems by means of a Single-objective Genetic Algorithm

Margaritis Basdavanos
School of Science & Technology
Hellenic Open University
Patras, Greece
mgbasdavan@yahoo.gr

Alexander Chatzigeorgiou
Department of Applied Informatics
University of Macedonia
Thessaloniki, Greece
achat@uom.gr

Abstract - Behavior and state allocation in object-oriented systems is a rather non-trivial task that is hard to master and automate since it is guided by conceptual criteria and therefore relies on human expertise. Since attributes and methods can be placed in the classes of a system in uncountable different ways, the task can be regarded as a search space exploration problem. In this paper we present our experience from treating this issue by a genetic algorithm, which in contrast to previous approaches, is aiming at single-objective optimization. The fitness function is based on a novel metric which ensures that optimization improves both coupling and cohesion. The approach has been implemented as an Eclipse plugin allowing the effortless experimentation on any system. The evaluation results indicate that the problem is suitable for single-objective genetic algorithms and that an optimal or near-optimal solution can be obtained within reasonable time.

Keywords - object-oriented design; coupling; cohesion; genetic algorithm

I. INTRODUCTION

The design quality of an object-oriented (OO) software system is usually assessed by quantifying internal features, such as coupling and cohesion, employing appropriate software metrics. Corrective and especially adaptive maintenance that has not been anticipated initially, might lead to a degradation of the original design in a software system, a phenomenon that is known as software ageing or decay [17]. In terms of metrics, software decay is easily observable when metric values worsen drastically over time.

The design of an object-oriented system relies heavily on the allocation of behavior and state to the system classes and this task is often presented as the most important objective of Object-Oriented Analysis and Design (OOAD) techniques. This process is inherently difficult, since various alternative solutions exist and there is no formal way of reaching the most efficient one. System methods and attributes, constituting behavior and state respectively, can be placed in system classes in uncountable different ways, forming a problem with several degrees of freedom. The process is usually guided by conceptual criteria (i.e., domain knowledge) or design principles [14] and heuristics [18] (e.g., keeping related behavior and data in one place). However, the choices that a designer makes, do not always lead to a system that is optimal or even close to optimal, in terms of coupling and cohesion. Moreover, even if the initial design is well structured, software decay, as already mentioned, might lead in the course of time to an inefficient placement of methods and attributes, lowering the changeability [4], comprehensibility, maintainability [2], testability and reusability [6] of the system.

The formulation of the above objective as a search problem in the space of the alternative arrangements of responsibilities (methods) and information (attributes) in the software units (classes) of an object oriented system can be faced as an optimization problem [5]. The goal is to distribute attributes and methods in a way that minimizes coupling and maximizes cohesion, thereby facilitating system maintenance. For a system with c classes and e entities (attributes and methods), the possible arrangements of entities in the classes are equal to c^e, a number that becomes extremely large even for medium-sized projects. Genetic Algorithms offer a reliable and robust non-linear search method for this kind of problems.

The proposed approach employs as fitness function the Entity Placement metric [20] which captures both coupling and cohesion using a common set of features for their evaluation. The fact that a single criterion is used enables the application of Single-Objective Genetic Algorithms (SOGA) guaranteeing that the process leads always to a single optimal or near-optimal solution that minimizes the value of the fitness function.

The proposed approach can be employed both at early stages of the analysis/design process when limited information about method/attribute dependencies is available but also at later stages when a detailed class diagram or even the source code is available to suggest possible improvements. Since the employed fitness function is by definition a measure of system coupling over cohesion, a single-objective GA can be used without the need to revert to arbitrary metric weightings or summations. The advantage of a single-objective over a multi-objective approach is that a unique solution is extracted, relieving the designer from the additional effort of inspecting various alternatives that correspond to different tradeoffs between the various objectives. Finally, it should be mentioned that the approach has been fully automated as an Eclipse-plugin which retrieves from source code all method/attribute dependencies and applies the genetic algorithm to extract the optimum solution.

The rest of the paper is organized as follows: Section II describes the use of Entity Placement as a fitness function, while the problem formulation is presented in Section III. Two case studies are analyzed in Section IV and the results are discussed in Section V. Section VI presents the implementation tool. Section VII provides an overview of the related work. Finally we conclude in section VIII.

II. ENTITY PLACEMENT AS A FITNESS FUNCTION

Since methods and attributes are the free variables which the designer of an object-oriented system, following either a systematic analysis and design methodology or his intuition, has to allocate to the individual classes, it is important to be able to characterize the quality of a given setting. The software engineering literature has recorded various design principles that should be followed [14] or design heuristics that should not be violated when taking design decisions [18]. Many of these principles and heuristics are inherently related to the central notions of coupling and cohesion signifying their importance in assessing the design quality of a given system. Consequently, the fitness function that we have employed for evaluating a possible solution in the problem of behavior and state allocation is based on coupling and cohesion.

Although these two measures, for which numerous metrics have been proposed, are strongly related, conventional metrics do not quantify coupling and cohesion with common terms or symbols. As a result, despite the fact that it would be tempting to define a ratio of cohesion over coupling, which the designer would seek to maximize, no such function has been successfully applied. The rest of this section describes a novel metric proposed in [20], which encompasses both coupling and cohesion and serves perfectly the goal of single-objective optimization.

A. Measures of similarity between system entities

The constituents of an OO system, within the context of our study are methods and attributes which can be placed in several classes. Methods can access attributes and methods of the class that they belong to directly and also attributes and methods of other classes through references. Conversely, attributes can be accessed directly from methods of the class that they belong to and also from methods of other classes that have reference to that class.

For each entity (attribute/method) it is possible to define a set of entities that it accesses (if it is a method) or a set of entities that it is accessed from (if it is an attribute). The formation of the set is straightforward; however, the following rules should be taken into account [20]: Since references are essentially a pipeline to the state or behavior of another class, attributes that are references to classes of the system should not be considered as entities nor added to the entity sets of other entities. However, the accessed methods/attributes through reference should be taken into account in the formation of the corresponding entity sets.

Since library classes are fixed from the programmer's perspective and are not subject to refactoring, access to attributes/methods of classes outside the system boundary (e.g., library classes) should not be taken into account.

Similarly to the entities, it is also possible to define the entity set of a class C containing all attributes (except references to system classes) that belong to class C and all methods that belong to class C.

It is reasonable to assume that the similarity between an entity and a class is high when the number of common entities in their entity sets is large. The similarity between two entities can be obtained employing the Jaccard similarity coefficient between the corresponding entity sets. For two sets A and B the Jaccard similarity coefficient is defined as the cardinality of their intersection divided by the cardinality of their union.

The distance between two entities is obtained by the Jaccard distance between their entity sets. For two sets A and B the Jaccard distance is defined as:

$$distance(A,B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|} = 1 - \frac{|A \cap B|}{|A \cup B|} = 1 - similarity(A,B)$$

In the context of our problem, let e be an entity of the system, C a class of the system and S_x the entity set of entity or class x. The distance between an entity e and a class C is calculated as follows:

Definition 1, if the entity e does not belong to the class C, the distance is the Jaccard distance of their entity sets:

$$distance(e,C) = 1 - \frac{|S_e \cap S_C|}{|S_e \cup S_C|}, \text{ where } S_C = \bigcup_{e_i \in C} \{e_i\}$$

Definition 2, if the entity e belongs to the class C, e is not included in the construction of S_C:

$$distance(e,C) = 1 - \frac{|S_e \cap S'_C|}{|S_e \cup S'_C|}, \text{ where } S'_C = \bigcup_{e_i \in C, e_i \neq e} \{e_i\}$$

In this way, it is ensured that all distance values range over the interval [0, 1].

B. Global measure of entity placement

In a well designed system that adheres to the principle of grouping behavior with related data, the distances of the entities belonging to a class (inner entities) from the class itself should as small as possible (high cohesion). At the same time the distances of the entities not belonging to a class (outer entities) from that class should be as large as possible (low coupling). This can be ensured by considering for each class the ratio of average inner to average outer entity distances. For each class, the closer this ratio to zero is, the safer it can be concluded that inner entities have correctly been placed inside the class and outer entities to other classes. A formula that provides the above information for a class C is given by [20]:

$$EntityPlacement_C = \frac{\dfrac{\sum_{e_i \in C} distance(e_i, C)}{|entities \in C|}}{\dfrac{\sum_{e_j \notin C} distance(e_j, C)}{|entities \notin C|}}$$

where e denotes an entity of the system.

A global measure of how well entities have been placed in the classes is obtained by the weighted metric for the entire system which considers the number of entities in each class:

$$EntityPlacement_{System} = \sum_{C_i} \frac{|entities \in C_i|}{|all\ entities|} EntityPlacement_{C_i}$$

The lower the value of this metric is, the better the placement of entities since entities not belonging to a class have a large distance from it, and entities belonging to a class exhibit a small distance. In a way the numerator expresses inter-class coupling (which should be minimized) and the denominator expresses intra-class cohesion (which should be maximized). Therefore, the EntityPlacement metric can perfectly serve as a fitness function for a genetic algorithm exploring alternatives of behavior and state allocation.

III. INVESTIGATING THE OPTIMUM PLACEMENT BY MEANS OF GENETIC ALGORITHMS

A. Genetic Algorithms - Terminology

Genetic Algorithms (GAs) are search techniques, based on the principles of evolution and natural selection, used in computing to find exact or approximate solutions to optimization and search problems [9]. Potential solutions are encoded appropriately (usually as binary or integer strings) called *chromosomes*. A GA explores the search space usually by producing offsprings with combination of two parents, while the exploration begins with a random population of chromosomes or individuals. The size of a chromosome is equal to the number of problem variables, the elements of chromosomes are named *genes*, the different values that a gene can take are called *alleles*, while the positions of genes in the chromosome are called *loci*.

Individuals of a given population are evaluated employing a suitable fitness function. This function plays the role of the environment in which individuals evolve and its value guides the reproduction process [15]. Depending on the problem the fitness function can return larger or smaller values for the better individuals. To produce new individuals and to mimic natural methods of random change GAs also apply genetic operators [11], such as crossover and mutation in the chromosomes. The best individuals from a generation that are selected for reproduction are chosen according to a selection strategy. In our study we employed the binary tournament method where from two chromosomes drawn at random from the population, the chromosome with the highest fitness score is selected (according to a predefined selection probability) as parent.

B. Problem Formulation

The genetic algorithm, applied to the problem of finding an optimum allocation of methods and attributes in a given system, can be outlined in the following five phases:

1. Design of the chromosome

The chromosome constitutes the heart of GA. For an object-oriented system each chromosome consists of integer genes, while its size is equal to the total number $\boldsymbol{e}$ of the system entities. Therefore, the loci of chromosomes range in the interval [0, $\boldsymbol{e}$-1]. The alleles express the coded values of the model classes and vary in the interval [0, $\boldsymbol{c}$-1], where $\boldsymbol{c}$ is the total number of system classes. For example, an allele of 5 indicates that the corresponding entity (method or attribute) should be placed in the class coded as 5.

2. Selection of fitness function

The employed fitness function accepts a potential solution and returns the value of the Entity Placement metric that indicates how good that solution is relative to other possible solutions. Since Entity Placement expresses a ratio that resembles coupling over cohesion, the lower the value, the better the solution.

3. Determination of configuration parameters

The next step is to specify the parameters of the GA, i.e., the crossover rate $\boldsymbol{p_c}$, the mutation rate $\boldsymbol{p_m}$, and the selection rate $\boldsymbol{p_s}$ for binary tournament, that expresses the selection probability of the best individual. Since we used SOGA, a conservative value of 50% [7] for the crossover rate has been selected. As our GA uses mild elitism (i.e., the fittest individual is always selected for the next generation) a relatively high mutation rate is necessary for optimum performance. A number of authors suggest a mutation rate based on the length of chromosome and concretely 5/length (5 mutations per chromosome) [13]. Others [3] obtained better results with a lower rate of 1/length. Since the length of the chromosome in our case study is even smaller (38 genes), we adopted a much lower mutation rate of 1/(2*length) for more stable results.

Regarding the binary tournament, the selection rate has been set to 80%, which means that the fitter chromosome is selected 80% of the time.

Generally further increasing the crossover and mutation rate increase the run time, but does not appear to lead to important improvements in the quality of solutions. Moreover, further increasing the selection probability decreases the diversity of population and it can bias the exploration in concrete solutions.

4. Creation of initial population

The initial population is created randomly and consists of $\boldsymbol{n}$ individuals with the previously defined characteristics. The determination of the ideal size of a population is a challenging and important task. For traditional GAs sizes from 30 up to 80 have been proposed [10] or between 1.2 to 3.5 times the total variables of the problem. According to our experimental results, a population size in the range of 1.2 to 2 times the system entities appears to be sufficient.

5. Evolution of population and termination conditions

The initial population evolves through the generations (evolutions or cycles) until the solution satisfies a certain value for the fitness function. The evolution might also terminate when a specified number of cycles has been reached. An alternative criterion is the convergence rate of the chromosomes (usually a rate of 95%).

IV. CASE STUDIES

A. Hypothetical Model

To test the effectiveness of the proposed approach, a hypothetical model that includes three classes ($\boldsymbol{c}$=3) *A*, *B*

and *C*, with six entities (***e***=6, 3 attributes and 3 methods), is employed. Method *m1* accesses the attribute *y* of class *B*, method *m2* the attribute *z* of class *C* and method *m3* the attribute *x* of class *A*, as it appears in the diagram of Fig. 1. Arrows indicate the attributes that each method accesses.

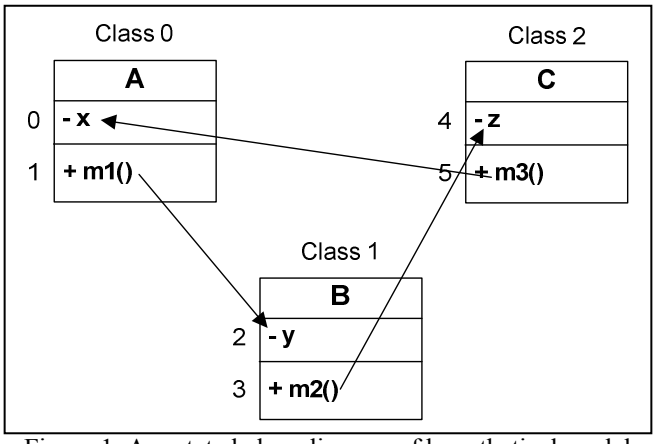

Figure 1. Annotated class diagram of hypothetical model

According to the previous analysis, the encoding of the shown allocation corresponds to the following chromosome:

0	0	1	1	2	2
0	1	2	3	4	5

Figure 2. Initial solution

For this initial solution the value of the fitness function is equal to 1.333. After the application of the GA the following optimal solution is obtained:

0	1	1	2	2	0
0	1	2	3	4	5

Figure 3. A final optimal solution

The fitness value for the optimal solution is zero, proving that the proposed approach is sound, since methods are placed in the same classes with the attributes that they access. For the given example, it is obvious that this allocation minimizes coupling and maximizes cohesion and thus the solution is indeed the best one. The corresponding annotated class diagram is shown in Fig. 4.

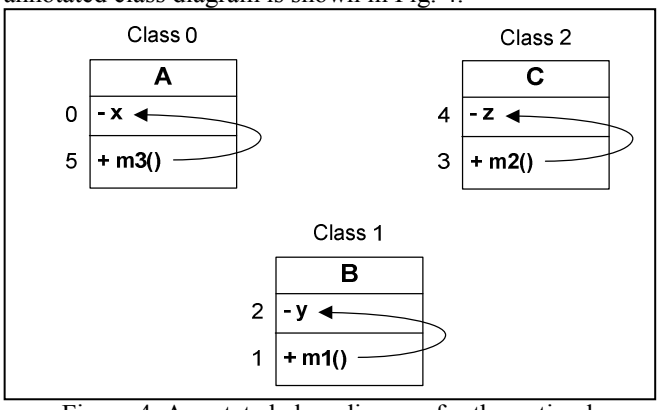

Figure 4. Annotated class diagram for the optimal hypothetical model

The initial population for this case was set to 12 (two times the number of entities). With a crossover percentage p_c=50%, mutation percentage p_m=8.33% and selection rate p_s=80%, the GA resulted almost always in an optimal solution with fitness value zero, a fact that appears also in the following indicative diagrams of chromosome convergence rate and fitness value through generations (Fig. 5 and 6). The average run time of the GA was approximately 60ms (Pentium IV, 3.2 GHz, 3 GB RAM).

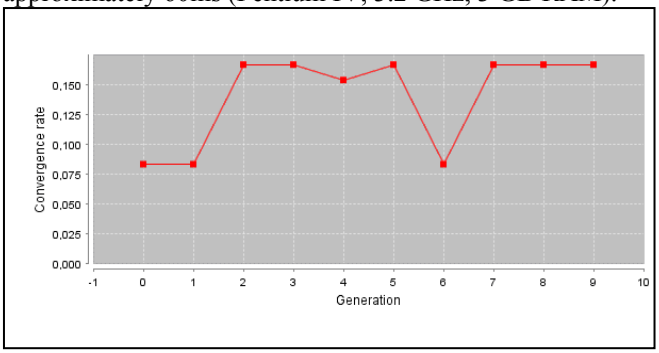

Figure 5. Convergence rate by generation

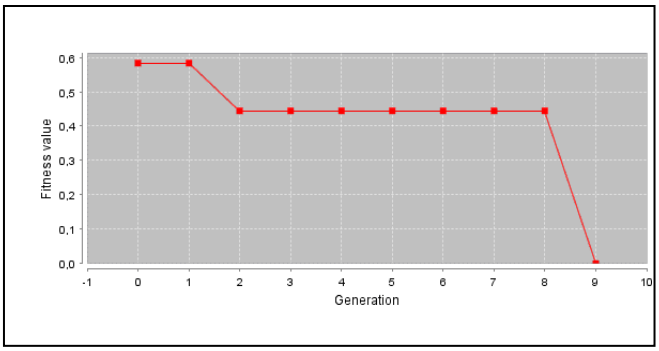

Figure 6. Fitness value by generation

B. *Order Management*

To test the effectiveness of the proposed approach on a larger system (which however can be inspected for the resulting quality) an object-oriented system resulting from the application of the ICONIX methodology has been selected. The design refers to an order management system and the evaluation of the optimal allocation of methods and attributes by the genetic algorithm can be tested against the architecture resulting from the application of the analysis and design methodology. The system includes ***c***=6 domain classes and ***e***=38 entities (attributes / methods), creating a relatively large search space of $c^e = 6^{38}$ possible allocations. The 38 entities are coded in the integer interval [0, 37] and the 6 classes in the integer interval [0, 5], using this coding scheme in the application of the GA. The value of the fitness function for the design resulting from the application of the ICONIX methodology is equal to 0.714.

In our experiments we used as maximum number of generations 150 or 200 and the following set of genetic parameters: [**A**: p_c = 50%, p_m = 2.63%, p_s = 60%], [**B**: p_c = 50%, p_m = 1.32%, p_s = 80%]. Moreover a limit for the fitness value (lower than that of the initial system) was set as criterion of premature termination. In most cases the GA

led to near-optimal solutions with fitness values close or lower to that of the design resulting from the ICONIX methodology. The evolution of the fitness value for cases **A** and **B** are shown in Fig. 7 and 8 respectively.

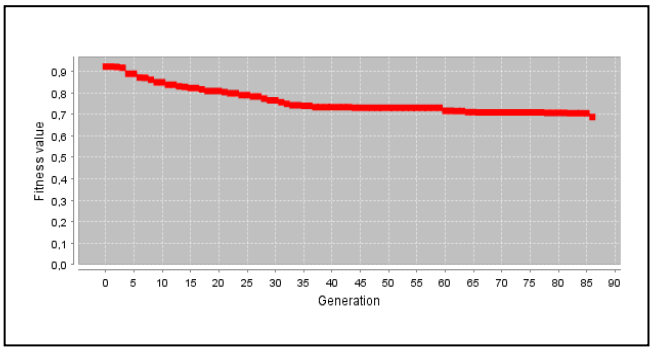

Figure 7. Fitness value by generation (parameters A)

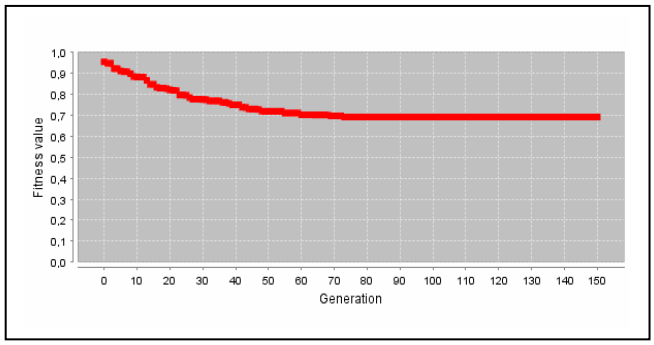

Figure 8. Fitness value by generation (parameters B)

From all experimental measurements it can be concluded that, for this case study, an actual limit for the fitness value exists, which appears to be near the value 0.690. Average run times per case, are presented in Table I.

Table I. Average run time

Gene-rations	p_c=50% p_m=2.63% p_s=60% Average run time (sec)	p_c=50% p_m=1.32% p_s=80% Average run time (sec)
150	15.7	13.5
200	20.6	18.0

(*On a Pentium IV, 3.2 GHz, 3 GB RAM)

V. DISCUSSION

Regarding the OrderManagement example, good enough solutions have been achieved before 100 or 150 generations depending on the configuration parameters. For the case study a crossover percentage of 50%, a mutation rate in the range 1% - 3% and a selection probability of 60% to 80%, generally optimize the performance of GA. With the above configuration the run times ranged from 13.5 sec to 15.7 sec for 150 generations and from 18 sec to 20.5 sec for 200 generations, times which are reasonable for the size of problem. Additionally it was observed that a further increase of crossover and mutation percentage increase the run time, without significant improvement in the resulting solutions.

The differences between the fitness values of the resulting solutions from the corresponding value of the initial system are also an indicator of the validity of the approach. The Entity Placement metric value for the best resulting system (0.687) is very close to that of the initial system (0.714). Since the system under study resulted from the application of the ICONIX methodology and thus the allocation of methods/attributes can be considered reasonable, it can be claimed that the genetic algorithm addresses the problem effectively. Many of the best solutions maintain half of the classes of the initial system intact, while other classes are split and one of the resulting fragments is incorporated into one of the existing classes. It should be noted that the genetic algorithm based allocation did not result in any "God" classes avoiding the excessive accumulation of functionality or data in certain classes.

VI. IMPLEMENTATION

All phases of the proposed approach have been implemented and integrated in an Eclipse plugin that is available upon request. The plugin allows the automatic extraction of all information required for the application of the genetic algorithm and then applies the algorithm to obtain the optimum solution. The information which is extracted by the Abstract Syntax Tree provided by Eclipse, includes the entity sets for all entities (attributes/methods), and the entity sets for all classes. The sets for each entity are not subject to change since they reflect the decisions that have been taken during analysis and design. The extracted information is then fed to the genetic algorithm that has been implemented employing the JGAP genetic algorithms framework [12]. The output consists in the coded solutions per generation with the corresponding fitness values, the final solution with its fitness value, the final sets of classes as well as the graphical representations of convergence and fitness values per generation.

VII. RELATED WORK

There are several approaches in the literature that treat the design of an object-oriented system as a multi-objective optimization problem in which the goal is to reassign methods and attributes to classes (either from scratch or by employing stepwise modifications in the form of refactorings [8]) in order to optimize the value of selected metrics, typically coupling and cohesion. The common denominator of these approaches is that the "independent" variables of a software design are its behavior, expressed by the methods that provide the functionality and its state, expressed by the attributes holding the system's information.

O'Keeffe and O'Cinneide [16] employ search algorithms, such as Hill Climbing and Simulated Annealing, to select refactorings in order to move through the space of alternative designs. The quality evaluation function that ranks the alternative designs is based on metrics from the QMOOD hierarchical design quality model [1].

The work of Seng et al. [19] employs genetic algorithms that suggest a sequence of Move Method refactorings which can improve the structure of system classes. For the definition of the fitness function the approach requires the specification of an arbitrary trapezoidal function for the normalization of certain metrics (such as WMC and NOM), a calibration run for optimizing each metric separately, and

the specification of weights. The required method/attribute dependencies are retrieved from UML models such as sequence diagrams. A direct consequence is that if the models are not completely specified, the resulting application of the GA will not reflect the actual state of the system under study.

In their methodology, Bowman et al. [3] used the class diagram of an object oriented system as entry in a Multi-Objective Genetic Algorithm (MOGA), which suggests possible improvements in the assignment of methods and attributes. The GA fitness function is based on multiple complementary measures of coupling and cohesion. The selected coupling and cohesion values have to be added to form a final class diagram coupling and cohesion measure, respectively. The choice of the more complex MOGA stems from the fact that it is very difficult to combine coupling and cohesion criteria into one unique function. The authors consider that it is a priori difficult for any designer to weigh different design criteria and therefore employ the Pareto based multi-objective algorithm. One of the key consequences of this decision is that a large number of alternative, non-dominated solutions can be recovered (solutions whose corresponding fitness function vectors are not dominated by other solution vectors) and therefore the final decision is left to the designer. It should be mentioned that the multi-objective GA with the selected Pareto approach is highly sensitive on the configuration parameters and thus careful tuning and experimentation is required. Finally, the computational cost for the application of MOGA even on a small-sized system was prohibitively large in terms of time.

The gradual identification of design problems and application of the corresponding refactorings (such as Move Method refactoring) [20], can also be regarded as a human-assisted exploration of the problem space in the search of an optimum allocation of method and attributes. However, it must be guided by designer decisions at each step.

VIII. Conclusions and Future Work

This paper presents an approach to assist the allocation of methods and attributes in an object-oriented system, a task which is inherently difficult to teach and apply effectively. The proposed treatment is based on single-objective genetic algorithms employing an Entity Placement metric as a fitness function. The optimal solution, expressed as an ideal allocation of system entities to classes, is the one that minimizes coupling and maximizes cohesion.

Two case studies have shown that the approach leads to optimal or near-optimal proposals from a variety of possible solutions that can be obtained within a reasonable number of generations and time. The encoding of the problem and the application of the GA has been implemented as an Eclipse-plugin facilitating the seamless application during design.

Future work includes the experimentation with larger systems in order to test the validity of the employed Entity Placement metric. Other important topics are related to the possibility of an effective interaction between the designer and the GA, the appropriate handling of inheritance hierarchies and the consideration of restrictions with regard to entities which according to domain knowledge should be placed in a certain class.

References

[1] J. Bansiya and C.G. Davis, "A hierarchical model for object-oriented design quality assessment", IEEE Trans. Software Eng., vol. 28, no. 1, pp. 4-17, Jan. 2002.

[2] A. Binkley and S. Schach, "Validation of the coupling dependency metric as a predictor of runtime failures and maintenance measures", Proc. 20th Int. Conf. Software Engineering (ICSE'98), April 1998, Kyoto, Japan, pp. 452-455.

[3] M. Bowman, L. Briand, and Y. Labiche, "Multi-objective algorithms to support class resposibility assignment", 23rd IEEE Int. Conf. on Software Maintenance (ICSM'07), October 2007, Paris, France, pp. 124-133.

[4] L. Briand, J. Wust, and H. Lounis, "Using coupling measurment for impact analysis in object-oriented systems", Proc. Int. Conf. on Software Maintenance (ICSM'99), September 1999, Oxford, England, UK, pp. 475-482.

[5] B. Bruegge and A. Dutoit, Object-Oriented Software Engineering, Prentice Hall, 2nd Edition, NJ, USA, 2004.

[6] S. Chidamber, D. Darcy, and C. Kemerer, "Managerial use of metrics for object-oriented software: An exploratory analysis", IEEE Trans. on Software Engineering, vol. 24, no 8, pp. 629-639, Aug. 1998.

[7] K. De Jong, "Learning with genetic algorithms: An overview", Machine Learning , 3 (3), pp. 121-138, 1998.

[8] M. Fowler, Refactoring: Improving the Design of Existing Code, Addison Wesley, USA, 1999.

[9] D. Goldberg, Genetic Algorithms in Search Optimization & Machine Learning, Addison Wesley, USA, 1989.

[10] J. Grefenstette and H. Cobb, "Genetic algorithms for tracking changing environments",5th Int. Conf. on Genetic Algorithms, Urbana, IL, USA, pp.523-530, June 1993.

[11] R. Haupt and S. Haupt, Practical Genetic Algorithms, Wiley-Interscience, 2nd Edition, NJ, USA, 2004.

[12] JGAP, http://jgap.sourceforge.net/, 2010

[13] M. Laumanns, E. Zitzler, and L.Thiele, "On the effects of archiving, elitism, a density based selection in evolutionary multi-objective optimization", Proc. 1st Int. Conf. on Evolutionary Multi-Criterion Optimization (EMO'01), March 2001, Zurich, Switzerland, pp.181-196.

[14] R. C. Martin, Agile Software Development: Principles, Patterns and Practices, Prentice Hall, 2003.

[15] Z. Michalewicz, Genetic Algorithms+Data Structures = Evolution Programs, 3d Edition, Springer, 1996.

[16] M. O'Keeffe and M. O'Cinnéide, "Search-based software maintenance", Proc. 10th European Conf. Software Maintenance and Reengineering (CSMR'06), March 2006, Bari, Italy, pp. 249-260.

[17] D. L. Parnas, "Software aging", Proc. 16th Int. Conf. on Software Engineering (ICSE'94), May 1994, pp. 279–287.

[18] A. J. Riel, Object-oriented design heuristics, Addison-Wesley, 1996.

[19] O. Seng, J. Stammel, and D. Burkhart, "Search-based determination of refactorings for improving the class structure of object-oriented systems", Proc. 8th Annual Conf. Genetic and Evolutionary Computation (GECCO'06), July 2006, Seattle, WA, USA, pp. 1909-1916.

[20] N. Tsantalis and A. Chatzigeorgiou, "Identification of move method refactoring opportunities", IEEE Trans. on Software Engineering, vol. 35, no. 3, June 2009, pp. 347-367.