Journal of Computing and Information Technology - CIT 19, 2011, 4, 275–283
doi:10.2498/cit.1002023

Solving the Class Responsibility Assignment Problem Using Metaheuristic Approach

Goran Glavaš and Krešimir Fertalj

Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia

Assigning responsibilities to classes is among first and arguably most important steps when creating object-oriented software design. This step depends greatly on human judgment and experience. In this paper our objective is to automatize assigning responsibilities to classes using metaheuristic optimization algorithms. Four different algorithms (simple genetic algorithm, hill climbing, simulated annealing and particle swarm optimization), using class coupling and cohesion metrics, were implemented and their results are compared. Implemented algorithms take semantically annotated responsibility dependency graph as input. This paper describes responsibility dependency graph, implemented algorithms and used coupling and cohesion metrics in detail. Paper also reports on a performed case study. Ultimately, based on results obtained from all implemented algorithms, conclusions on search landscape of class responsibility assignment problem are drawn.

Keywords: class responsibility assignment, genetic algorithm, hill climbing, simulated annealing, particle swarm optimization

1. Introduction

Obtaining good software design in general is a difficult task and creating object-oriented (OO) software design is a very complex process that includes several steps. Initial steps include recognizing key abstraction in problem domain (i.e. class candidates) and assigning responsibilities of a system to them [3]. Initial design, with responsibilities assigned to classes, serves as basis for applying more advanced OO mechanisms like inheritance, interfaces, abstract classes etc. Having responsibilities assigned and fine-tuning mechanisms like interfaces and inheritance applied experienced designer could, finally, add design patterns and possibly architectural styles to further improve design. However, poor assignment of responsibilities to classes cannot be fixed by inheritance or design patterns. Admittedly, there are well-described methodologies created to help recognize responsibilities of a system [13] as well as assign them to classes [6], but all of them rely on human reasoning.

In this paper we address class responsibility assignment as a search problem, making it fit for application of metaheuristic (MH) algorithms. Our objective is to compare different MH algorithms and investigate how suitable they are for solving class responsibility assignment problem. We use responsibility dependency graph (RDG) as a starting point for design creation opposed to much more often used approach of improving existing assignment of responsibilities to classes (i.e. OO design refactoring). Our work primarily focuses on domain model classes and assigning responsibilities on a higher level of abstraction.

Our work includes implementation of four different search algorithms: genetic algorithm (GA), hill climbing (HC), simulated annealing (SA) and particle swarm optimization (PSO). All algorithms use same class coupling and cohesion metrics in order to find optimal or near-optimal solution to class responsibility assignment problem. Implementation of several different MH algorithms (with different search characteristics) and comparing their results on a case study offers insight to the nature of search landscape.

The rest of the paper is structured as follows. Section 2 describes related work. Section 3 provides details on our representation of the problem, semantically annotated RDG and cohesion and coupling metrics used for evaluation. In Section 4 we describe all four implemented MH algorithms. Case study and performance results MH algorithms are provided in Section 5. Discussion of results and conclusions are drawn in Section 6.

2. Related Work

During the last decade there has been a significant tendency to formulate software engineering problems as search problems, making them suitable for metaheuristic search algorithms application. Large variety of problems, throughout all phases of software engineering project's lifecycle, have been formulated and dealt with as search problems [8, 10]. However, the amount of search-related work in the subfield of software architecture and design has been significantly lower than in other subfields (like testing and maintenance) of software engineering [11]. Also, most of the work considering design and architecture can be categorized as software maintenance or re-engineering [17]. There has been very little work reported on metaheuristic design synthesis from requirements.

In [18] genetic algorithm with fairly complex chromosome, encoding is used to synthesize object-oriented architecture of a system based on requirements given in the form of responsibility dependency graph. We follow this idea of RDG being an input for MH algorithms, but we expand RDG with semantic annotations. We argue that these annotations are not domain specific and therefore our approach can be applied in general. However, work in [18] is among the first focused on generating design from requirements and not refactoring existing design. Authors in [4] focus on class responsibility assignment problem and try to solve it using multi-objective genetic algorithm. Existing design is used as input and GA tries to improve the design considering different class cohesion and coupling metrics. Opposite to [18] and similar to [4], our approach focuses solely on class responsibility assignment problem, but we concentrate on creating initial OO design from requirements rather than refactoring existing design.

Most of the work in the field of search-based software design generation and refactoring implement solely genetic algorithms (or some similar evolutionary techniques) without providing argumentation on the choice of MH method [1, 4, 18]. In [14] authors report that HC outperforms GA on the problem of software module clustering. We notice significant similarities between software module clustering problem and class responsibility assignment problem. Hence, we implement four different search techniques and argue that simple GA is not the best choice for class responsibility assignment problem.

3. Class Responsibility Assignment as Search Problem

In order to make problem suitable for application of search-based optimisation algorithms, there are only two prerequisites required [11]:

1. The choice of the representation (i.e. encoding) of the problem
2. The definition of the fitness function.

We chose the simplest and most obvious way to represent the candidate solution (i.e. encoding). Each responsibility of a system is represented by one gene in the chromosome. Chromosomes are simply arrays of integer values where the position of the gene in the chromosome denotes the responsibility of a system and the gene's integer value denotes the class the responsibility is assigned to. Though simple, this encoding ensures that there are no empty classes in the design and that every responsibility is assigned to exactly one class. The choice of fitness function was, however, much less obvious. There are very few (if any) OO metrics that can be seen as standard and are generally accepted [17]. We chose multi-objective optimization, normalizing and combining three different coupling and cohesion measurements into a single aggregated fitness function.

3.1. Semantically Annotated Responsibility Dependency Graph

Similar to [18], we use responsibility dependency graph as input for MH algorithms. But instead of encoding dependencies into the chromosome, we use separate dependency matrix, keeping the chromosome structure as simple as possible. Semantically, we differentiate two types of responsibilities: data (DR) and functional (FR). These relate to attributes and methods of a class respectively, but on a higher level of abstraction (e.g. one functional responsibility might be implemented as several methods in the implementation phase). In order to provide more data for MH algorithms to work on helping them find better solutions, we further annotate RDG. Let us first define our notation that will be used throughout the rest of the paper. Let R be the set of all responsibilities, FR set of functional and DR set of data responsibilities in RDG. Let D be the set of all dependencies in RDG. Let C be the set of classes the responsibilities are assigned to, c being a single class. $R(c)$, $FR(c)$, $DR(c)$ then represent all responsibilities, functional responsibilities and data responsibilities assigned to a class c respectively. We introduce six different types of dependencies between responsibilities, giving each type different coupling significance. Data dependency $(DD(fr, dr) \mid fr \in FR \wedge dr \in DR)$ is a dependency between a functional responsibility and data responsibility (meaning function uses data). Dependencies between two functional responsibilities can be of four different types:

1. *simple call dependency* $(SCD(fr_1,fr_2) \mid fr_1,fr_2 \in FR)$ – source function simply initiates destination function.
2. *parametrized call dependency* $(PCD(fr_1,fr_2) \mid fr_1,fr_2 \in FR)$ – source function initiates destination function and sends required data.
3. *simple call waiting for result* $(SCR(fr_1,fr_2) \mid fr_1,fr_2 \in FR)$ – source function initiates destination function and uses the result of its execution.
4. *parametrized call waiting for result* $(PCR(fr_1,fr_2) \mid fr_1,fr_2 \in FR)$ – source function initiates destination function sending required data and then uses the result of its execution.

Sometimes good object oriented design means putting together (i.e. in the same class) semantically related responsibilities, though there are no direct dependencies between them. This improves understandability of the design, but it also degrades both cohesion and coupling aspects of the design. *Conceptual dependency* $(CD(dr_1, dr_2) \mid dr_1, dr_2 \in FR)$ was introduced in order to express such semantically related data responsibilities. Each dependency type has the value of unit coupling assigned to it. Unit coupling of a dependency $(uc(d) \mid d \in D)$ is a measure of coupling that single dependency introduces to the design. Unit coupling of two responsibilities is defined as follows:

$$uc(r_1, r_2) = \begin{cases} uc(d(r_1, r_2)) & \text{if } d(r_1, r_2) \in D \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

All of the described semantic annotations of a responsibility dependency graph serve as fine-tuning mechanisms for better assignment of responsibilities to classes.

3.2. Fitness Measurement

There are many different coupling and cohesion measures [5, 7]. Our approach is also driven by coupling and cohesion metrics, but we also introduce measurement for assignment of conceptually (semantically) related responsibilities. Three measures that we use (modularization quality, positive cohesion of methods and conceptual data cohesion) are explained in detail. Aggregated fitness function was designed as a linear combination of these individual measures.

Modularization quality (MQ) was first introduced in [9] for module clustering problem. It consists both of coupling and cohesion measures called *inter-connectivity* (coupling) and *intra-connectivity* (cohesion). Acknowledging significant similarities between module clustering problem and class responsibility assignment problem we use adjusted MQ. For a class c intra-connectivity is defined as follows:

$$A(c) = \frac{\sum\limits_{r \in R(c)} \sum\limits_{s \in R(c)} uc(r, s)}{uc_{average} |R(c)|^2} \tag{2}$$

For classes c_1 and c_2 inter-connectivity is defined as follows:

$$E(c_1, c_2) = \frac{\sum\limits_{r \in R(c_1)} \sum\limits_{s \in R(c_2)} uc(r, s)}{2uc_{average} |R(c_1)| |R(c_2)|} \tag{3}$$

Modularization quality is defined as difference between scaled averaged value of intra-connectivity over all classes and scaled averaged value of inter-connectivity over all pairs of classes:

$$MQ = a_1 \frac{\sum_{i=1}^{|C|} A(c_i)}{|C|} - a_2 \frac{\sum_{i=1}^{|C|-1} \sum_{j=i+1}^{|C|} E\left(c_i, c_j\right)}{\binom{|C|}{2}} \tag{4}$$

where a_1 and a_2 are scaling factors for expressing relative influence of cohesion and coupling respectively. The value of MQ is increased by increasing cohesion and lowering coupling of the design.

Positive cohesion of methods (PCOM) was inspired both by *lack of cohesion of methods* (LCOM) [7] and *tight class cohesion* (TCC) [5] measures. PCOM can be thought of as inverse of LCOM. While LCOM punishes solutions where methods of the same class do not use same class attributes, PCOM rewards solutions where functional responsibilities of the same class are dependent on the same data responsibilities of the class. Similar to TCC, PCOM of a class is defined as the percentage of pairs of functional responsibilities of the class with common usage of data responsibilities of the class. Let us define common data responsibility set for two functional responsibilities fr_1 and fr_2 of the same class c:

$$\begin{aligned} CDR\,(c,fr_1,fr_2) = \{&dr \in DR(c) \mid \\ &d(fr_1, dr) \in D \wedge d(fr_2, dr) \in D\} \end{aligned}$$

Positive cohesion of methods for a pair of functional responsibilities of the same class is defined as follows:

$$pcom(c,fr_1,fr_2) = \begin{cases} 1 & \text{if } |CDR\,(c,fr_1,fr_2)\,| > 0 \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

Positive cohesion of methods for a class is then defined as follows:

$$Pcom(c) = \frac{\sum\limits_{fr_1 \in FR(c)} \sum\limits_{fr_2 \in FR(c)} pcom(c,fr_1,fr_2)}{\binom{FR(c)}{2}} \tag{6}$$

PCOM of a solution design is then averaged value of *Pcom* values for all classes.

As described earlier, in the semantically annotated RDG there may exist special conceptual dependency between two data responsibilities expressing that they semantically relate (i.e. both are part of the same real world abstraction). We introduce *conceptual data cohesion* (CDC) as a measure that rewards solutions which group conceptually dependent data responsibilities together (i.e. in the same class). We define conceptual connection between data responsibilities as a transitive relation: if responsibility A is conceptually connected to responsibility B and responsibility B is conceptually connected to responsibility C, then A is also conceptually connected to C, but with a lower weight value assigned to that dependency since connection is not direct. Conceptual data cohesion for a class c is defined as follows:

$$Cdc(c) = \frac{\sum\limits_{dr_1 \in DR(c)} \sum\limits_{dr_2 \in DR(c)} cdc(c, dr_1, dr_2)}{\binom{DR(c)}{2}} \tag{7}$$

where $cdc(dr_1, dr_2)$ has value 1 if dr_1 and dr_2 are directly connected, value 0 if there is no connection (not even transitive) between dr_1 and dr_2 and value between 0 and 1 (proportional to distance) if dr_1 and dr_2 are remotely conceptually connected. CDC of a solution design is then averaged value of *Cdc* values for all classes in the solution.

4. Implementation

In order to compare performance and applicability of different MH algorithms on the class responsibility assignment problem, we created the *Class Responsibility Deployer* (CRD) tool. CRD provides support for drawing semantically annotated RDG, executing MH algorithms and presenting best found solution vector in the form of class diagram. Here we describe four implemented MH algorithms in more detail. Performance results of each implemented algorithm are given in Section 5. We chose to create our own implementations of aforementioned algorithms rather than use existing frameworks.

4.1. Genetic Algorithm

Valid changes in the solution vector consist of moving responsibilities between classes (i.e. changing integer values of genes in the chromosomes). We implemented both extreme possibilities for crossover operator: 1-point crossover and uniform crossover. We used roulette wheel as selection operator. The most obvious mutation, considering our encoding of candidate solutions, was to randomly reassign responsibilities to different classes. All classes had equal probability of responsibility being reassigned to them. Mutation was applied on a gene level. Mutation rate was experimented with.

Our choice of the population size was proportional to the size of the search landscape. We set population size to be 4 times number of responsibilities in the RDG. Two special case chromosomes - having only one class and each having responsibility in its own class were always added in the initial population. GA performed better with 1-point crossover than with uniform crossover. Considering mutation, GA performed best with gene-level mutation probability of $1/(length)$ where *length* is the number of genes in chromosome. GA also performed better with elitism supported.

4.2. Hill Climbing

Hill climbing starts from a randomly chosen chromosome. In every iteration of the algorithm the neighbourhood of the current chromosome is searched in order to find fitter solution. Definition of search neighbourhood is problem specific. We define neighbourhood of a chromosome as all chromosomes that are exactly "one mutation away" (i.e. compared to current solution, exactly one responsibility is assigned to a different class). We implemented two main approaches for choosing the fitter neighbour: *next ascent hill climbing* where the first fitter neighbour is selected and *steepest ascent hill climbing* where entire neighbourhood is examined and the neighbour solution that gives the greatest increase in fitness is selected. Hill climbing ends when there is no fitter neighbour in the neighbourhood. The biggest shortcoming of the hill climbing is that it finishes in the local optimum closest to the random initial point. This can be resolved by repeatedly restarting the algorithm at different starting points. This is called *multiple-ascent hill climbing* (MAHC). We implemented both next ascent MAHC and steepest ascent MAHC. Next ascent MAHC had, naturally, significantly lower execution time, but it was also able to find fitter solutions than steepest ascent MAHC.

4.3. Simulated Annealing

Simulated annealing algorithm is similar to hill climbing but it allows less fit neighbour to be selected for next iteration. This way, simulated annealing reduces the possibility of ending in the closest local optimum of the starting point. The probability of accepting less fit neighbour is bigger in the earlier iterations of algorithm (traversing through search landscape more freely) and decreases progressively. It is also inversely proportional with the decrease in fitness that neighbour presents compared to the current solution. Probability of accepting less fit neighbour is defined as follows:

$$p = e^{\frac{fn-fc}{T}} \tag{8}$$

4.4. Particle Swarm Optimization

Particle swarm optimization (PSO) algorithm was inspired by social behaviour of some animal species (e.g. flock of birds). Every specimen (particle) in the group (flock, herd, pack, ...) can benefit from the discoveries and previous experience of all other members during the search for the food [12].

PSO algorithm starts with a population of particles where each particle's initial position is a random solution in the search landscape. In each iteration of the algorithm every particle changes its position based on both cognitive and social influence. Cognitive influence is determined by the best solution found by particle itself. Social influence is determined by the best solution found by all particles in the neighbourhood. The definition of neighbourhood is problem specific.

We adjusted general PSO algorithm to fit our encoding of class responsibility assignment problem. In every iteration each particle changes its current solution according to its own best found solution and best found solution of all particles in its neighbourhood. Two probabilities are calculated and applied on gene level: local probability is the probability of accepting gene's value from the particle's best solution; global probability is the probability of accepting gene's value from the best solution of the entire neighbourhood. Local probability is proportional to cognitive coefficient and difference in fitness between particle's best found solution and particle's current solution. Global probability is proportional to social coefficient and difference in fitness between neighbourhood's best found solution and particle's current solution. In case that both gene values (particle's best solution gene value and neighbourhood's best solution gene value) are accepted, the applied value gets chosen according to the ratio of cognitive coefficient and social coefficient. We implemented two different versions of the algorithm regarding definition of neighbourhood. Global neighbourhood PSO defines entire swarm as one neighbourhood. Local neighbourhood PSO defines neighbourhood of a particle to be only a smaller part of the entire swarm. Global neighbourhood PSO displayed faster convergence, but often led to nonsatisfactory local optimum. Local neighbourhood PSO with neighbourhood size of 5-10% of swarm's overall size generally resulted with significantly better solutions. PSO algorithm achieved best results with cognitive coefficient value of 0.8 and social coefficient value of 0.2.

5. Case Study

The objective of the case study was to determine up to what level are implemented MH algorithms able to recognize main abstractions in problem domain (i.e. class candidates) and assign responsibilities to them. Case study also served to evaluate how relevant chosen fitness measures are for good OO design. Completely objective analysis of the results would, however, require optimal reference model for comparison. The optimality of OO design is very hard (if possible at all) to define, hence all the different (and often contradictory) existing evaluation metrics. But our goal was

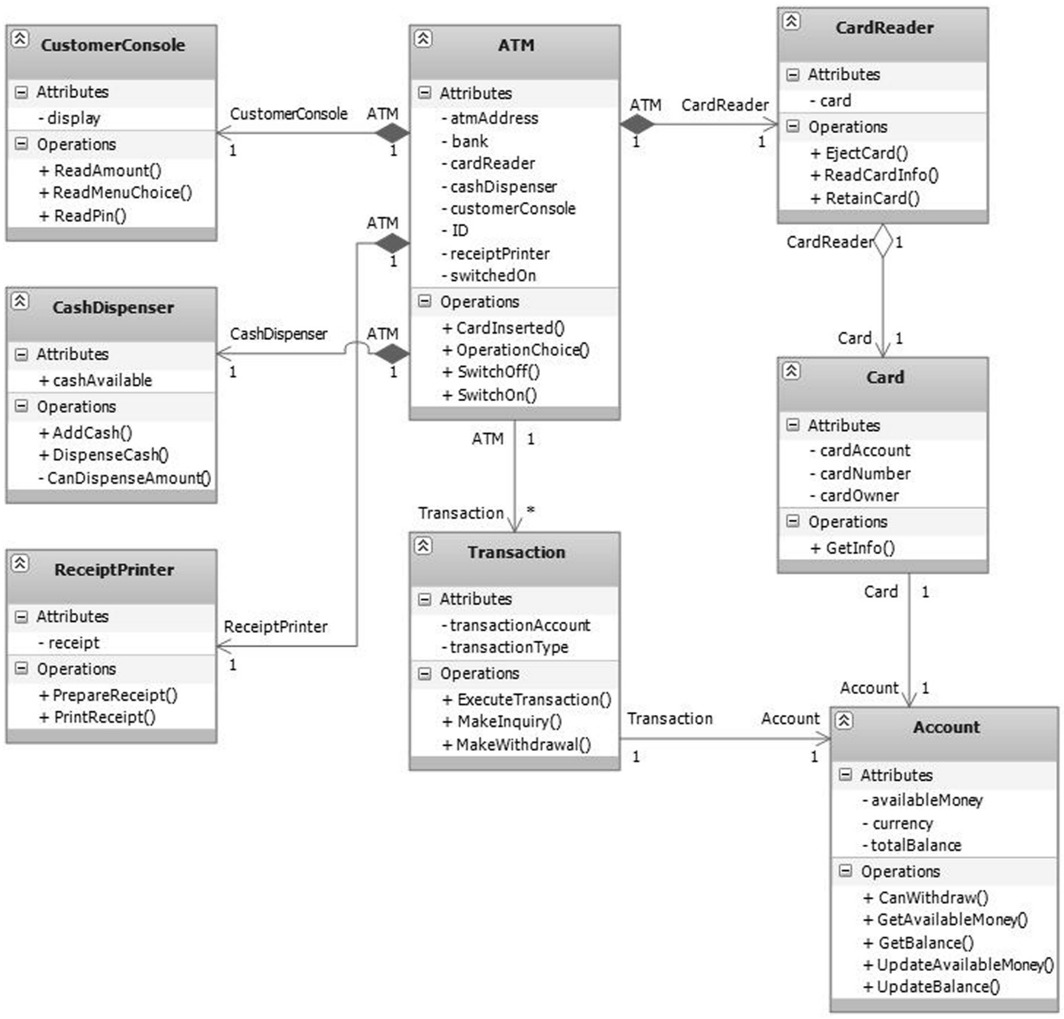

Figure 1. ATM Simulator reference model

not set to formalize optimal OO design, but to investigate whether MH algorithms can produce design close to what human judgment would consider satisfactory. Therefore, our reference model was known, human created satisfactory design. Comparing best-fitted solutions considering chosen metrics to reference model can then serve to evaluate how appropriate the choice of metrics was as well.

In order to increase objectivity of our approach, it was necessary to choose a domain model independent of our research. We selected ATM Simulation domain model [2] since it comes with recognized responsibilities and dependencies from which we can form RDG. However, we considered entire ATM Simulation model to be too large for initial case study, so we selected logical subset of responsibilities and adequate subdesign for reference model (Figure 1). Our reference model contained 8 classes having 44 responsibilities (20 data and 24 functional) assigned to them.

In order to keep generality, maximal possible number of classes in the solution has to be equal to number of responsibilities which makes the size of search space considerably large – 44^{44}. To confirm that search problem is not trivial and search space is considerably large, we performed random search of 500000 iterations along with four implemented MH algorithms.

All four algorithms performed better than random search (i.e., passed the "sanity check"). MAHC and MASA managed to find significantly better solutions than PSO. Simple GA performed worst. MAHC and MASA were able to recognize main abstractions (i.e. classes) and assign most of the responsibilities "correctly" (Figure 2). PSO recognized some of the abstractions, while GA was able to group some dependent responsibilities together, but hardly recognized classes from the reference model. Table 1 describes best found solution for each algorithm (including random search): fitness, total number of classes, number of recognized classes, number of correctly assigned responsibilities and number of incorrect assignments. By human judgment, solutions produced by

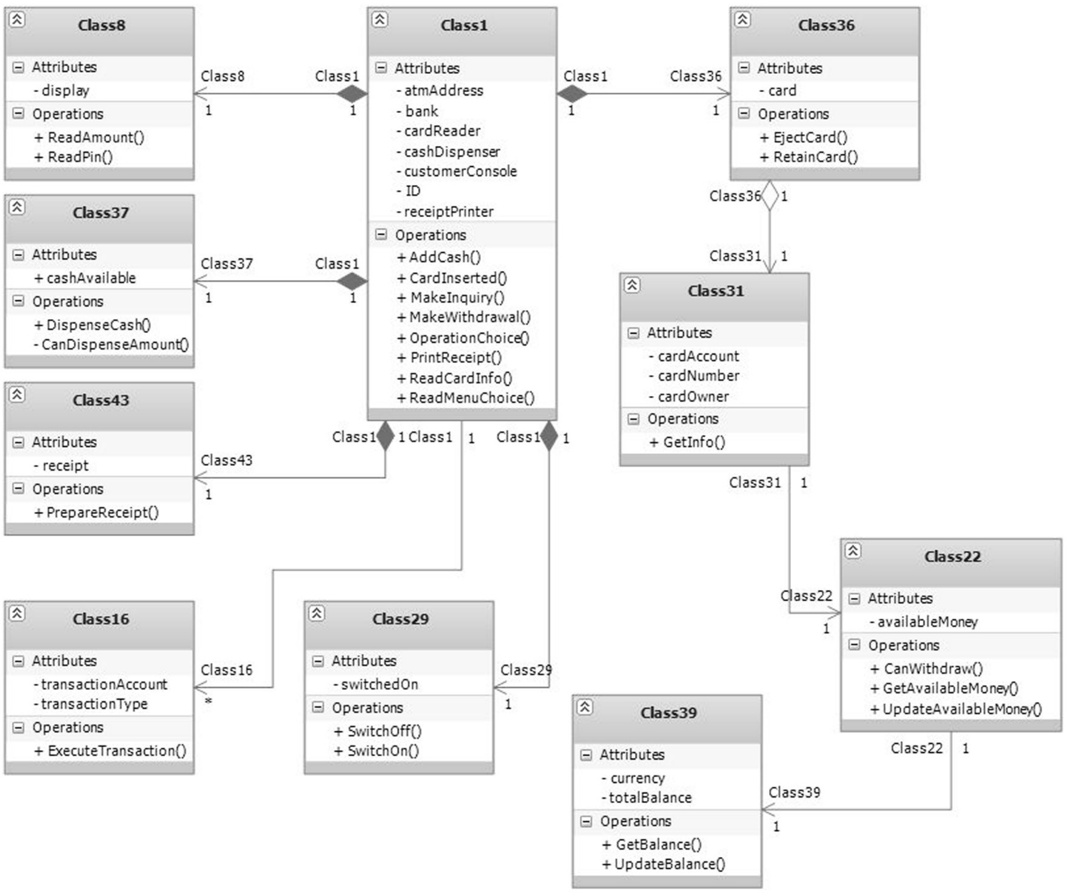

Figure 2. The best design generated by MH algorithm

MAHC and MASA that had better fitness were also recognized as closer to satisfactory design. This indicates that metrics constructing aggregate fitness function were well selected. Still, fitness of reference design (1.77) is slightly smaller than fitness of best designs produced by MASA and MAHC. However, human judgment evaluates reference design to be slightly better than those solutions. This means there is room for further improvement of selected fitness metrics.Correlation between good fitness and close to satisfactory design was increased when intra-connectivity component of MQ was reduced, i.e. $a_1 \ll a_2$ in (4). This is understandable since both other measures (PCOM and CDC) target cohesion as well, so cohesion was initially over-expressed by aggregated fitness function.

Algorithm	*Fitness*	*Number of classes*	*Classes recognized*	*Correct assignments*
GA	0.87	28	3	6
MAHC	1.83	9	7	41
MASA	1.85	10	8	38
PSO	1.34	15	5	12
Rand. S.	0.50	26	1	2

Table 1. Performance results of algorithms

6. Discussion

In the discussion, we focus on analyzing the search landscape of class responsibility assignment problem. Conclusions drawn regarding the nature of search landscape provide an explanation of results produced by implemented metaheuristic algorithms. Let us observe fitness of a chromosome as aggregation of fitnesses of individual genes. Fitness of a single gene in our encoding depends not only on the value of that particular gene, but also on values of all genes that represent dependent responsibilities (i.e. fitness contribution of responsibility r being assigned to class c is determined by assignment of all responsibilities dependent on r and of all responsibilities r is dependent on). An NK landscape is a function defined on encodings of fixed length and is characterized by two parameters: n for the overall number of genes and k for the neighborhood size. For each gene, k neighbour genes influence its fitness contribution [16]. There has been research showing that with increasing k, crossover-based algorithms are outperformed by mutation-based algorithms [15]. Though the neighbourhood size k is not constant for all genes, search landscape of class responsibility assignment can be considered close to NK-landscape problem. This explains why simple crossover-based GA is outperformed by mutation-based algorithms like HC and SA. Furthermore, HC and SA were executed in multi-ascent manner, escaping its local nature to some extent. This is also the reason why GA with uniform crossover performed even worse than GA with single-point crossover. There are, however, reports of more robust GAs performing better than hill climbers on a wide variety of NK-landscape problems [16]. Such GAs should also be applied to class responsibility assignment problem. PSO also recombines genes from two chromosomes (best individual solution and best global solution), but it is more mutation-based than GA, thus performing better than GA, but worse than MAHC and MASA.

Regarding evaluation metrics, our research leads to a conclusion that solely coupling and cohesion metrics are not enough to produce sound designs. One of important characteristics of good OO design is understandability. However, to produce designs that are human understandable, some semantics must be provided as input as well. We introduced conceptual dependencies as a way to mark semantically related concepts. Admittedly, this can be considered designing itself. Though often considered in related work, we find usage of MQ for class responsibility assignment problem doubtful since we obtained better designs minimizing influence of its cohesion component. It is very hard to point out, and even harder to formalize in the form of measurements, all aspects of good OO design. All this implicates that partially automated creation of OO design is a much more likely approach than fully automated one.

7. Conclusion and Future Work

This paper presented automated approach to creating initial OO design from scratch, using metaheuristic search optimization algorithms to identify classes and assign responsibilities to them. Our case study showed that some metaheuristic algorithms are capable of producing designs very close to what human judgment would consider satisfactory. We believe that good designs can hardly be obtained in an automated way, using only structural measures and that some semantics must be provided as input too. Next steps in our research will further explore appropriate measures reflecting good OO design, but will also focus on applying more advanced features of OO design, such as inheritance, abstract classes and interfaces using MH algorithms.

References

[1] M. Amoui, S. Mirarab, S. Ansari and C. Lucas, A genetic algorithm approach to design evolution using design pattern transformation, *International Journal of Information Technology and Intelligent Computing*, vol. 1, no. 1, pp. 235–245, 2006.

[2] R. Bjork, An example of Object-oriented Design: An ATM Simulation, Department of Mathematics and Computer Science, Gordon University, `http://www.math-cs.gordon.edu/courses/cs211/ATMExample/`, 2004.

[3] G. Booch, R. Maksimchuk, M. Engle, B. Young, J. Conallen and K. Houston, Object-oriented analysis and design with applications, Addison-Wesley Professional, 2007.

[4] M. Bowman, L. C. Briand and Y. Labiche, Multi-objective genetic algorithm to support class responsibility assignment, *Software Maintenance, 2007. ICSM 2007. IEEE International Conference on*. IEEE, pp. 124–133, 2007.

[5] L. C. Briand, J. W. Daly and J. Wüst, A unified framework for cohesion measurement in object-oriented systems, *Empirical Software Engineering*, Springer, vol. 3, no. 1, pp. 65–117, 1998.

[6] B. Bruegge and A. H. Dutoit, *Object-oriented software engineering*, Prentice Hall, 2000, vol. 553.

[7] S. R. Chidamber and C. F. Kemerer, A metrics suite for object oriented design, *Software Engineering, IEEE Transactions on*, vol. 20, no. 6, pp. 476–493, 1994.

[8] J. Clarke, J. J. Dolado, M. Harman, R. Hierons, B. Jones, M. Lumkin, B. Mitchell, S. Mancoridis, K. Rees, M. Roper et al., Reformulating software engineering as a search problem, *Software, IEE Proceedings-*, IET, 2003, vol. 150, no. 3, pp. 161–175.

[9] D. Doval, S. Mancoridis and B. S. Mitchell, Automatic clustering of software systems using a genetic algorithm, *Software Technology and Engineering Practice, 1999. STEP'99. Proceedings*, IEEE, 1999, pp. 73–81.

[10] M. Harman, The current state and future of search based software engineering, *2007 Future of Software Engineering*, IEEE Computer Society, 2007, pp. 342–357.

[11] M. Harman, S. A. Mansouri and Y. Zhang, Search based software engineering: A comprehensive analysis and review of trends techniques and applications, *Department of Computer Science, Kingòs College London, Tech. Rep. TR-09-03*, 2009.

[12] J. Kennedy and R. Eberhart, Particle swarm optimization, *Neural Networks, 1995. Proceedings., IEEE International Conference on*, IEEE, 1995, vol. 4, pp. 1942–1948.

[13] C. Larman and P. Kruchten, *Applying UML and patterns*, Prentice Hall PTR, 2002.

[14] B. S. Mitchell and S. Mancoridis, On the automatic modularization of software systems using the Bunch tool, *IEEE Transactions on Software Engineering*, Published by the IEEE Computer Society, 2006, pp. 193–208.

[15] P. Merz and B. Freisleben, On the effectiveness of evolutionary search in high-dimensional NK-landscapes, *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, IEEE, 1998, pp. 741–745.

[16] M. Pelikan, Analysis of estimation of distribution algorithms and genetic algorithms on NK landscapes, *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, ACM, 2008, pp. 1033–1040.

[17] O. Räihä, A survey on search-based software design, *Computer Science Review*, Elsevier, 2010.

[18] O. Räihä, Applying genetic algorithms in software architecture design, *University of Tampere Department of Computer Sciences*, Citeseer, 2008.

Received: June, 2011
Accepted: November, 2011

Contact addresses:

Goran Glavaš
Faculty of Electrical Engineering and Computing
University of Zagreb
Zagreb, Croatia
e-mail: `goran.glavas@fer.hr`

Krešimir Fertalj
Faculty of Electrical Engineering and Computing
University of Zagreb
Zagreb, Croatia
e-mail: `kresimir.fertalj@fer.hr`

Goran Glavaš is a PhD student at the Department of Electronics, Microelectronics, Computer and Intelligent Systems at the Faculty of Electrical Engineering and Computing, University of Zagreb. He graduated and received his B.Sc. and M.Sc. degrees in computing from the same institution. His professional and scientific interest is in intelligent software systems, especially in natural language processing and search optimization algorithms.

Krešimir Fertalj is a full professor at the Department of Applied Computing at the Faculty of Electrical Engineering and Computing, University of Zagreb. His professional and scientific interest is in computer-aided software engineering, complex information systems and in project management. He has written over a hundred scientific and professional publications and participated in conferences locally and abroad. Fertalj is member of ACM, IEEE, PMI, and of Croatian Academy of Engineering.