A Metrics-Based Evolutionary Approach for the Component Selection Problem

Conference Paper · April 2009

DOI: 10.1109/UKSIM.2009.32 · Source: IEEE Xplore

1 author:

Andreea Vescan
Babeş-Bolyai University
66 PUBLICATIONS **186** CITATIONS

SEE PROFILE

A Metrics-based Evolutionary Approach for the Component Selection Problem

Andreea Vescan
Faculty of Mathematics and Computer Science
Babes-Bolyai University
Cluj-Napoca, Romania
Email: avescan@cs.ubbcluj.ro

Abstract

Component-Based Software Engineering is concerned with the assembly of pre-existing software components that leads to software systems that respond to client specific requirements. Component selection and component systems assembly have become two of the key issues involved in this process.

There may be different alternative components that can be selected (to obtain a configuration), each coming at their own set of offered requirements. Our goal is to select the best existing component alternatives. In order to attain this goal, the evaluation of the components is a necessity. As a result in this direction, software metrics are very useful being a mean to quantify those attributes considered important for the system that will built.

We formulate the problem as multiobjective, considering different metrics values. The approach is an evolutionary computation technique. The experiments and comparisons (considering different criteria and various number of individuals and iterations) shows the importance of multicriteria-based solution.

1. Introduction

Component-Based Software Engineering (CBSE) is concerned with composing, selecting and designing components [1]. Software systems are built by assembling components already developed and prepared for integration. The main objective of CBSE is to obtain a more efficient development with shorter development times and better quality products: using an existing component, rather than developing a new one is often the quickest and cheapest method of meeting any given need.

In this paper, we address the problem of (automatic) component selection. Informally, our problem is to select a set of components from available component set which can satisfy a given set of requirements. When we select a component, there may be different components that offer similar functionalities. We aim at a selection approach that take into consideration some attributes of the components, initially established. Thus, results the necessity to evaluate these components. As a result in this direction, software metrics are very useful being a mean to quantify those attributes considered important for the system that will built.

The paper is organized as follows: Section 2 starts with the problem formulation. Section 3 presents the used metrics. The proposed approach (that uses an evolutionary algorithm) is presented in Section 4. In Section 5 some experiments and comparisons are performed. Related work on Component Selection is discussed in Section 6. We conclude our paper and discuss future work in Section 7.

2. Component Selection Problem

A formal definition of the problem is as follows. Consider SR the set of final system requirements (target requirements) as $SR = \{r_1, r_2, ..., r_n\}$ and SC the set of components available for selection as $SC = \{c_1, c_2, ..., c_m\}$. Each component c_i can satisfy a subset of the requirements from SR denoted $SR_{c_i} = \{r_{i_1}, r_{i_2}, ..., r_{i_k}\}$. In addition $cost(c_i)$ is the cost of component c_i.

The goal is to find a set of components Sol in such a way that every requirement r_j ($j = \overline{1,n}$) from the set SR can be assigned a component c_i from Sol where r_j is in SR_ci, taking into consideration various criteria.

3. Component assembly evaluation using metrics

As already mentioned before, the problem of selection of a component from a set of available components is not unique. In general, there may be different alternative components that can be selected, each coming at their own set of offered requirements. Our goal is to select the best existing component alternatives. In order to attain this goal, the evaluation of the components is a necessity. As a result in this direction, software metrics are very useful being a mean to quantify those attributes considered important for the system that will built. Thus, for each attribute we define some relevant metrics. Based on the interpretation of the

measurements results obtained, we decide what component best satisfies the system need.

In our study, we have considered for the system that will build three attributes: cost, reusability and functionality and in the following we define metrics for these attributes.

Cost Metric. The cost of a component (C), is defined as the overall cost of acquisition and adaptation of that component. We aim at finding a solution that minimize the total cost of the system.

Reusability Metrics. Reusability is of great importance to any software product. In what follows we will establish metrics that emphasize some aspects related with reusability of a component. An exhaustive presentation of these metrics is not the purpose of this article, we consider some reusability metrics closely tied to our problem.

In [2] Hoek et al. proposed metrics to asses service utilization in component assemblies. We use in this article two of these metrics: Provided Services Utilization Metrics(PSU), defined as the ratio of services provided by the component which are actually used (Equation 1 - left side) and Required Services Utilization (RSU) Metric, witch is similar with PSU metric but for required services (Equation 1 - right side).

$$PSU_X = \frac{P_{actual}}{P_{total}} \qquad RSU_X = \frac{R_{actual}}{R_{total}} \quad (1)$$

where:

- P_{actual} = number of services provided by component X that are actually used by other components;
- P_{total} = number of services provided by component X;
- R_{actual} = number of services required by component X that are actually provided by the assembly;
- R_{total} = number of services required by component X.

A value near to 1 for PSU metric indicates a poor reusability, while a value near to 0 indicates that the component could be reusable if the system evolve and adds other requirements. For RSU metric, a value near to 1 is a good reusablity indicator.

Functionality Metric. Functionality metric for a component represents the ratio between the number of required services provided by the component and the number of system required services. A component with metric value near to 1 offer several functionalities of the system.

4. Proposed approach description

Evolutionary algorithms are a part of evolutionary computing, which is a rapidly growing area of artificial intelligence. Inspired by Darwin's theory of evolution - Genetic Algorithms (GAs) are computer programs which create an environment where populations of data can compete and only the fittest survive, sort of evolution on a computer. They are well known suitable approaches for optimization problems.

The approach presented in this paper uses principles of evolutionary computation and multiobjective optimization [3].

There are several ways to deal with a multiobjective optimization problem. In this paper the weighted sum method [4] is used.

Let us consider we have the objective functions f_1, f_2,..., f_n. This method takes each objective function and multiplies it by a fraction of one, the "weighting coefficient" which is represented by w_i. The modified functions are then added together to obtain a single cost function, which can easily be solved using any method which can be applied for single objective optimization.

Mathematically, the new function is written as:

$$\sum_{i=1}^{n} w_i f_i,$$

where: $0 \leq w_i \leq 1$ and $\sum_{i=1}^{n} w_i = 1$.

4.1. Solution representation

A solution (chromosome) is represented as a string of size equal to the number of requirements from SR. The value of $i-th$ gene represent the component satisfying the $i{-}th$ requirement. The values of these genes are not different from each other (which means, same component can satisfy multiple requirements).

4.2. Genetic operators

The genetic operators used are crossover and mutation. Each of them is presented below.

4.2.1. Crossover operator. We use a simple one point crossover scheme. A crossover point is randomly chosen. All data beyond that point in either parent string is swapped between the two parents.

For example, if the two parents are: $parent_1$ = [0 4 1 0 7 1] and $parent_2$ = [0 4 8 0 7 7] and the cutting point is 3, the two resulting offspring are: $offspring_1$ = [0 4 1 0 7 7] and $offspring_2$ = [0 4 8 0 7 1].

4.2.2. Mutation operator. Mutation operator used here consist in simply exchanging the value of a gene with another value from the allowed set. In other words, mutation of $i-th$ gene consists in allocating a different component in order to satisfy the requirement i.

For instance, if we have the chromosome $parent_1$ = [0 4 1 0 7 1] and we chose to mutate sixth gene, then a possible offspring can be $offspring_1$ = [0 4 1 0 7 7].

4.3. Algorithm description

In a steady-state evolutionary algorithm one member of the population is changed at a time. The best chromosome (or a few best chromosomes) is copied to the population in the next generation. Elitism can very rapidly increase performance of GA, because it prevents losing the best found solution to date. A variation is to eliminate an equal number of the worst solutions, i.e. for each "best chromosome" carried over a "worst chromosome" is deleted.

The general pseudocode of the evolutionary algorithm used in this paper is given in the Algorithm 1.

Algorithm 1 Evolutionary algorithm

```
Population Initialization;
for ( t = 1 to nEvolutionCount) do
    for ( k=0 to lstPopulationCount; k += 2)) do
        Randomly choose two individuals, p1 and p2;
        OneCutPointCrossover for p1 and p2, resulting
        child c1 and child c2;
        Mutation(c1); Mutation(c2);
        Memorize in c1 and c2 the best individual from p1
        and c1 and from p2 and c2;
        Replace the two worst individuals form population
        with c1 and c2.
    end for
end for
```

5. Experiments

A short and representative example is presented in this section. Starting for a set of six requirements and having a set of ten available components, the goal is to find a subset of the given components such that all the requirements are satisfied.

The set of requirements $SR = \{r_0, r_1, r_2, r_3, r_4, r_5\}$ and the set of components $SC = \{c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9\}$ are given.

Table 1 contains for each component the provided services (in terms of requirements of the final system).

Table 1. Requirements elements of the components in SC

	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9
r_0	√		√	√						√
r_1					√				√	
r_2		√				√			√	
r_3	√						√			
r_4						√	√	√		√
r_5		√					√	√		√

Table 2 contains the values of the metrics.

Table 2. Metrics

Comp.	PSU	RSU	F	C
c_0	0.66	0.50	0.33	0.08
c_1	0.50	0.60	0.33	0.07
c_2	1.00	1.00	0.16	0.06
c_3	0.25	0.20	0.16	0.09
c_4	0.50	0.00	0.16	0.06
c_5	0.50	0.60	0.33	0.14
c_6	1.00	1.00	0.50	0.15
c_7	0.50	0.33	0.33	0.14
c_8	0.50	0.50	0.33	0.07
c_9	0.50	0.50	0.50	0.14

The parameters used by the evolutionary approach are as follows: mutation probability: 0.7; crossover probability: 0.7; and number of different runs: 100.

We have performed two different experiments: first considering only two objectives and then considering all objective but with different population sizes and different number of generations.

5.1. Experiments with two objectives

The first type of experiments considered only two objectives: first experiment considers only Functionality and Cost, and the second experiment considers PSU and RSU.

5.1.1. Functionality and Cost criteria. Considering only two criteria (Functionality and Cost) the best obtained solution has the cost=0.28, the functionality equal to 0.99 (the fitness value is 0.635), and the representation: $[2\ 8\ 8\ 6\ 6\ 6\ ]$.

The evolution of the fitness function is given in Figure 1.

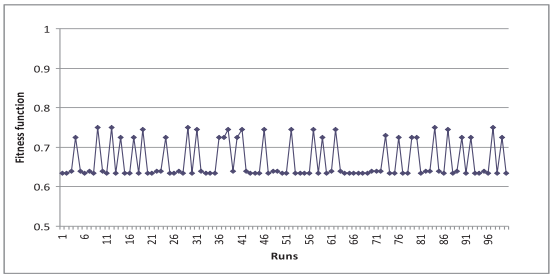

Figure 1. The evolution of fitness function ((best value) over the all 100 runs with Functionality and Cost criteria

5.1.2. PSU and RSU criteria. Considering only two criteria (PSU and RSU) the best obtained solution has the best fitness value is 1.495 (0.5*1.66(PSU)+0.5*1.33(RSU)), the cost is 0.29, and the representation is: $[0\ 8\ 8\ 0\ 7\ 7\ ]$.

The evolution of the fitness function is given in Figure 2.

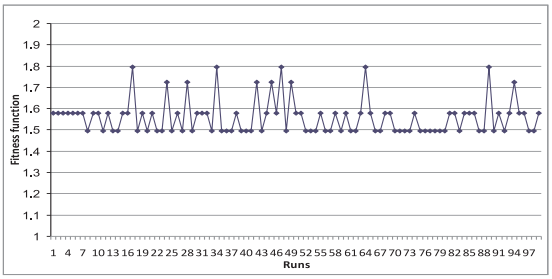

Figure 2. The evolution of fitness function (best value) over the all 100 runs with PSU and RSU criteria

5.2. Experiments with four objectives

The second type of experiments considered all four objectives: Functionality, Cost, PSU, RSU, and the population size and the number of iterations changes.

5.2.1. Experiment 1. For the first experiment we considered the following parameters: population size is 10 and number of iterations is 10.

The evolution of the fitness function is given in Figure 3.

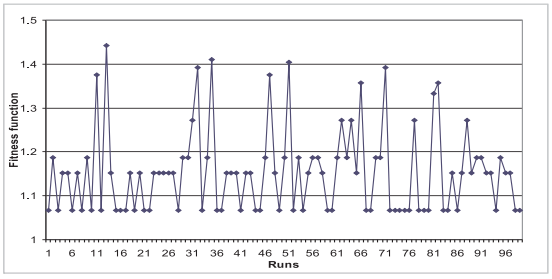

Figure 3. The evolution of fitness function (best value) over the all 100 runs, considering 10 individuals and 10 iterations and with all four metrics.

5.2.2. Experiment 2. For the second experiment we considered the following parameters: population size is 20 and number of iterations is 20.

The evolution of the fitness function is given in Figure 4.

5.2.3. Experiment 3. For the third experiment we considered the following parameters: population size is 50 and number of iterations is 50.

The evolution of the fitness function is given in Figure 5.

5.2.4. Experiment 4. For the fourth experiment we considered the following parameters: population size is 100 and number of iterations is 100.

The evolution of the fitness function is given in Figure 6.

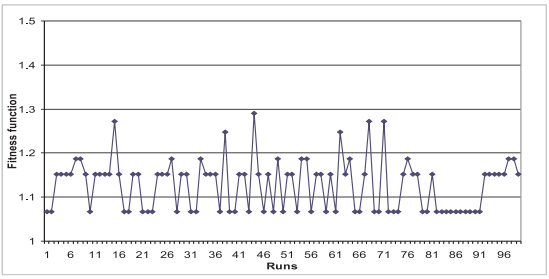

Figure 4. The evolution of fitness function (best value) over the all 100 runs, considering 20 individuals and 20 iterations and with all four metrics.

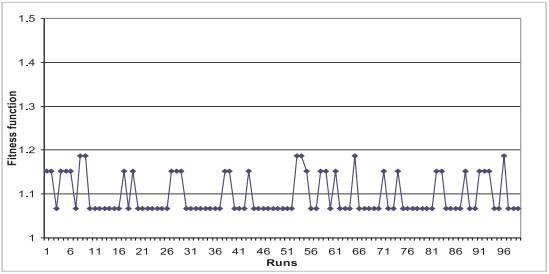

Figure 5. The evolution of fitness function (best value) over the all 100 runs, considering 50 individuals and 50 iterations and with all four metrics.

While compared with the previous experiments we noted that we are getting a lower number of different solutions while cumulation the results obtained in all the 100 runs, but the quality of these solutions is improving much more while compared with first experiment and the second one.

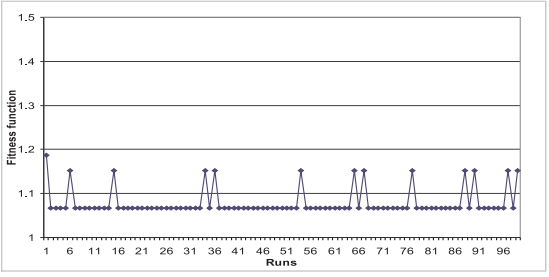

Figure 6. The evolution of fitness function (best value) over the all 100 runs, considering 100 individuals and 100 iterations and with all four metrics.

6. Related work

Component selection methods are traditionally done in an architecture-centric manner. An approach was proposed in

[5]. The authors present a method for simultaneously defining software architecture and selecting off-the-shelf components. They have identified three architectural decisions: object abstraction, object communication and presentation format. Three type of matrix are used when computing feasible implementation approaches. Existing methods include OTSO [6] and BAREMO [7].

Another type of component selection approaches is built around the relationship between requirements and components available for use. In [8] the authors have presented a framework for the construction of optimal component systems based on term rewriting strategies. By taking these techniques from compiler construction, especially optimizing code generation, they have developed an algorithm that finds a cost-optimal component system.

Paper [9] proposes a comparison between a Greedy algorithm and a Genetic Algorithm. The discussed problem considers a realistic case in which cost of components may be different. The selection function from the Greedy approach take into consideration both number of provided (offered requirements) of the components and the cost of the component.

Another type of component selection approaches is built around the relationship between requirements and components available for use PORE [10] and CRE [11]. The goal here is to recognize the mutual influence between requirements and components in order to obtain a set of requirements that is consistent with what the market has to offer.

In relation to existing component selection methods, our approach aims to achieve goals similar to [12], [13]. The [12] approach considers selecting the component with the maximal number of provided operations. The algorithm in [13] consider all the components to be previous sorted according to their weight value. Then all components with the highest weight are included in the solution until the budget bound has been reached.

7. Conclusion and future work

CBSE is the emerging discipline of the development of systems incorporating components. A challenge is how to assemble components effectively and efficiently.

Component Selection Problem has been investigated in this paper. We have proposed an evolutionary approach. In relation to existing approaches we have considered various metrics values of the involved components.

We intend to extend our approach by specifying and proving the compatibility between two connected components. The protocol for each provided operations of a component have to be specified and included into the composition process.

A future work will discuss the current proposal using a real case study. New conditions probably will be imposed.

8. Acknowledgement

This material is partially supported by the Romanian National University Research Council under award PN-II no. ID 2412/2009.

References

[1] I. Crnkovic, *Building Reliable Component-Based Software Systems*, M. Larsson, Norwood, MA, USA: Artech House publisher, 2002.

[2] A. v.d. Hoek, E. Dincel, N. Medvidovic, *Using Service Utilization Metrics to Assess and Improve Product Line Architectures*, Proceedings of the 9th IEEE International Software Metrics Symposium . New York, NY, USA: IEEE Computer Society, pp. 298–308, 2003.

[3] C. Grosan, *A comparison of several evolutionary models and representations for multiobjective optimization*, ISE Book Series on Real Word Multi-Objective System Engineering. New York, NY, USA: Nova Science, 2005.

[4] Y. Kim and O. L. deWeck, *Adaptive weighted-sum method for bi-objective optimization: Pareto front generation*, Structural and Multidisciplinary Optimization. New York, NY, USA: MIT Strategic Engineering Publications, pp. 149–158, 2005.

[5] E. Mancebo and A. Andrews, *A strategy for selecting multiple components*, Proceedings of the 2005 ACM symposium on Applied computing. New York, NY, USA: ACM, pp. 1505–1510, 2005.

[6] J. Kontio, *OTSO: A Systematic Process for Reusable Software Component Selection*, Technical report, University of Maryland. Maryland: Elsevier Science Publishers B. V., 1995.

[7] A. L. Tello and A. ón Gómez-Pérez, *BAREMO: how to choose the appropriate software component using the analytic hierarchy process*, Proceedings of the 14th international conference on Software engineering and knowledge engineering. New York, NY, USA: ACM, pp. 781–788, 2002.

[8] L. Gesellensetter and S. Glesner, *Only the Best Can Make It: Optimal Component Selection*, Electron. Notes Theor. Comput. Sci.. Amsterdam, The Netherlands: Elsevier Science Publishers B. V., pp. 105–124, 2007.

[9] N. Haghpanah and S. Moaven and J. Habibi and M. Kargar and S. H. Yeganeh, *Approximation Algorithms for Software Component Selection Problem*, Proceedings of the 14th Asia-Pacific Software Engineering Conference. Washington, D.C.: IEEE Computer Society, pp. 159–166, 2007.

[10] C. Alves and J. Castro, *Pore: Procurement-oriented requirements engineering method for the component based systems engineering development paradigm*, Proceedings of the Int'l Conf. Software Eng. CBSE Workshop. New York, NY, USA: ACM, 1999.

[11] C. Alves and J. Castro, *A systematic method for cots component selection*, Brazilian Symposium on Software Engineering. Rio De Janeiro, Brazil: ACM, 2001.

[12] M. R. Fox and D. C. Brogan and P. F. Reynolds, *Approximating component selection*, Proceedings of the 36th conference on Winter simulation. Washington, D.C.: IEEE Computer Society, pp. 429–434, 2004.

[13] P. Baker and M. Harman and K. Steinhofel and A. Skaliotis, *Search Based Approaches to Component Selection and Prioritization for the Next Release Problem*, Proceedings of the 22nd IEEE International Conference on Software Maintenance. Washington, D.C.: IEEE Computer Society, pp. 176–185, 2006.