A Novel Component Identification Approach Using Evolutionary Programming

Aurora Ramírez, José Raúl Romero and Sebastián Ventura
Dept. of Computer Science and Numerical Analysis, University of Córdoba
Rabanales Campus, 14071 Córdoba, Spain
{i72raqua, jrromero, sventura}@uco.es

ABSTRACT

Component identification is a critical phase in software architecture analysis to prevent later errors and control the project time and budget. Obtaining the most appropriate architecture according to predetermined design criteria can be treated as an optimization problem, especially since the appearance of the Search Based Software Engineering, and its combination with bio-inspired metaheuristics. In this work, an evolutionary programming (EP) algorithm is used to identify components, based on a novel and comprehensible representation of software architectures.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search -*Heuristic methods*

General Terms

Algorithms, Software design

Keywords

Component-based architecture, Search Based Software Engineering, Evolutionary Programming

1. INTRODUCTION

Architectural analysis places an important role in current software developments. It is mainly considered a human-centered decision process, where the abilities and prior experiences of software engineers have a marked influence on the end product quality and reusability. Therefore, during the high level analysis, components identification is a critical approach for dealing with complex systems [1], allowing to identify the different system elements, as well as their functionalities and interactions.

Recently, the appearance of SBSE (*Search Based Software Engineering*) [2] brings a new perspective for solving specific problems in Software Engineering through search and optimization approaches. Focusing on architecture optimization [3], aspects like the architecture definition, the quality attributes to be measured and the global objective (refactoring, deployment, etc.) reflects a variety of applications.

GECCO'13 Companion, July 6–10, 2013, Amsterdam, The Netherlands.
ACM 978-1-4503-1964-5/13/07.

In this paper we propose a novel evolutionary programming algorithm for software architecture optimization from analysis models. In a more precise way, a novel encoding of component-based software architectures closest to the expert domain comprehension, a fitness based on design concepts and specific genetic operators are presented.

2. ALGORITHM DESIGN

A component-based architecture can be described as a set of three elements: *components*, defined as a cohesive group of classes working together to satisfy its expected behaviour; *interfaces*, identified from relationships between classes belonging to different components that exchange services (required by one and provided by the other); and *connectors*, the linkage between a pair of required-provided interfaces.

A representation close to the expert domain has been selected, resulting in a trade-off between performance and comprehensibility. The hierarchical composition of these artefacts allows the translation into a tree structure (see Figure 1).

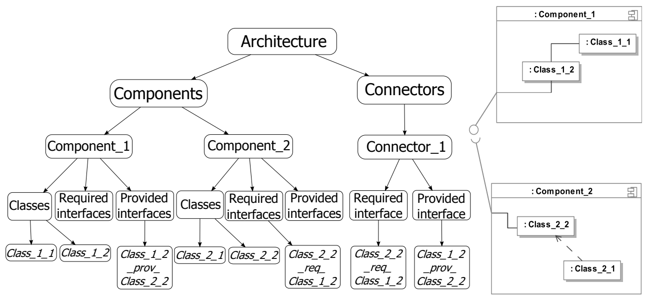

Figure 1: Genotype and phenotype

The initialization process begins with a random distribution of classes into a random number of components. Then, candidate interfaces and connectors are identified.

After their creation, individuals must be evaluated. Cohesion, the degree to which a component performs a well defined functionality, and coupling, related to the interdependence between components, are well-known design criteria. The distribution of classes among components is also important, as architectural solutions tend to look for balanced components in terms of size and inner complexity. Thus, these concepts are translated into quantifiable measures used to define the fitness function, f (see Equation 1).

$$coh_i = \frac{w_{c1}}{(w_a + w_d + w_{ac} + w_g) \cdot (n_{cl} - 1)} \cdot \Big[w_a \cdot n_a + w_d \cdot n_d + $$
$$+ w_{ac} \cdot n_{ac} + w_g \cdot n_g \Big] + w_{c2} \cdot \frac{1 - n_{gr}}{n_{cl} - 1}$$
$$cop = \frac{1}{R_{max}} \cdot \frac{2 \cdot R \cdot (C-2)!}{C!}; R = \sum_{i=1}^{C} \sum_{j=i+1}^{C} \max r_{i,j}^{k}$$
$$cv = \frac{\sigma}{\mu}$$
$$f = w_{coh} \cdot \frac{\sum_{i=1}^{n} coh_i}{n} + w_{cop} \cdot (1 - cop) + w_{cv} \cdot \left(\frac{8 - cv}{8} \right) \tag{1}$$

The global cohesion is calculated as the average cohesion of each component (coh_i), reflecting the strength of their internal structures. It considers a weighted sum of relationships among classes, based on the number of different types of UML relationships: associations (w_a, n_a), dependencies (w_d, n_d), aggregations and compositions (w_{ac}, n_{ac}), and generalizations (w_g, n_g). The presence of unconnected clusters of classes (n_{gr}) is penalized, since it might imply internal dispersion. n_{cl} is the number of internal classes.

Coupling (*cop*) concerns the relationships among classes belonging to different components, based on the number of combinations of C components. It varies between 0 (components only related through interfaces) and 1 (all components are mutually connected). R accumulates a penalty rate for each pair of linked components based on the strongest relationship between them ($\max r_{i,j}^{k}$). R_{max} represents the worst situation, i.e. all components are related with the rest by means of the strongest relationship.

The coefficient of variation, *cv*, provides a normalized measure of the component size dispersion. μ and σ represent the mean and standard deviation of the component internal size, respectively. In opposition to *coh*, *cv* and *cop* must be minimized, so arithmetical transformations using its maximum values are realised in order to maximize f.

Finally, each individual in the population is selected to act as parent, generating a new solution. A probabilistic roulette with five mutators is applied in order to obtain offsprings. Each one represents an architectural transformation: *add* and *remove* components, *split* and *merge* them or move classes from one component to another one. The replacement strategy establishes a competition between each parent and its offspring, so only the best individual survives.

3. EXPERIMENTATION

The complete approach has been written in Java using the Datapro4j library [1] and JCLEC framework [4]. To analyse the performance and accuracy of this proposal, 30 executions were performed over three diverse problems, i.e. architectural specifications. In absence of other proposals to compare with, a random search (RS) has been also performed. Table 1 shows the average of fitness values of the best solutions found. The parameter configuration used was: 100 individuals as population, 100 generations, 2-8 components, $w_{coh} = 0.3$, $w_{coupl} = 0.4$ and $w_{cv} = 0.3$.

In general terms, the EP algorithm obtains better solutions in all problem instances. Special attention must be placed in the first instance, where all solutions obtained achieve the minimum value for the coupling measure. Even when RS is able to find a set of components with similar sizes, the rest of measures are significantly worse. As for the EP approach, a more appropriate trade-off between size dispersion and the rest of measures has been achieved, which mainly benefits the cohesion. Finally, the EP algorithm is able to evolve and keep architectures with different number of components and connectors during the search.

[1]http://www.uco.es/grupos/kdis/datapro4j

Table 1: Results for EP and RS algorithms

		NekoHTML	AquaLush	Datapro4j
EP	Fitness	0.7216	0.6483	0.4690
	Cohesion	0.1762	0.1430	0.0586
	Coupling	0.0000	0.1411	0.5162
	CV	0.8333	1.0175	1.1218
RS	Fitness	0.6198	0.4640	0.3316
	Cohesion	-0.0390	-0.2510	0.0541
	Coupling	0.1420	0.3694	0.9522
	CV	0.3129	0.3460	0.1008

The Wilcoxon signed-rank statistical test has demonstrated that the EP algorithm performs significantly better than RS with 90% confidence.

4. CONCLUSIONS

This paper presents a novel approach for the identification of component-based architectures from analysis models. The proposed encoding uses trees structures, similar to the one used by the underlying specification models, which brings the approach closer to the software architect. Specific genetic operators simulating architectural transformations and a fitness function inspired in cohesion and coupling concepts conform the core of the evolutionary search.

Experimentation has shown very promising results in terms of cohesion and coupling. Furthermore, our approach can generate and evolve individuals with different number and configurations of components and connectors, showing a flexible handling of software architectures.

Acknowledgments

Work supported by the Ministry of Science and Technology, project TIN2011-22408, and FEDER funds.

5. REFERENCES

[1] C. Szyperski, *Component Software: Beyond Object-Oriented Programming.* Boston, USA: Addison-Wesley Longman Publ. Co., 2nd ed., 2002.

[2] M. Harman, S. A. Mansouri, and Y. Zhang, "Search Based Software Engineering: Trends, Techniques and Applications," *ACM Comput. Surv.*, vol. 45, no. 1, pp. 11:1–61, 2012.

[3] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, and I. Meedeniya, "Software architecture optimization methods: A systematic literature review," *IEEE Trans. on Software Engineering*, no. 99, pp. 1–26, 2012.

[4] S. Ventura, C. Romero, A. Zafra, J. A. Delgado, and C. Hervás, "JCLEC: a java framework for evolutionary computation," *Soft Comput.*, vol. 12, no. 4, pp. 381–392, 2007.