Multi-Objective Genetic Synthesis of Software Architecture

Outi Räihä
Department of Software Systems
Tampere University of Technology
FI-33101 Tampere, Finland
+358-50-5342813
outi.raiha@tut.fi

Kai Koskimies
Department of Software Systems
Tampere University of Technology
Tampere, Finland
kai.koskimies@tut.fi

Erkki Mäkinen
School of Information Sciences, Computer Science
University of Tampere, Tampere, Finland
erkki.makinen@uta.fi

ABSTRACT
A possible approach to partly automated software architecture design is the application of heuristic search methods like genetic algorithms. In order to take into account conflicting quality requirements, the use of Pareto optimality is proposed. This technique is studied in the presence of two central quality attributes of software architectures, modifiability and efficiency. The technique produces a palette of architecture proposals, and has been implemented and evaluated using an example system. The results demonstrate that Pareto optimality has potential for producing a sensible set of architectures in the efficiency-modifiability space.

Categories and Subject Descriptors
I.2.8 **[Artificial Intelligence]**: Problem Solving, Control Methods, and Search, D.2.11 **[Software Engineering]** Software Architectures, D.2.2 **[Software Engineering]** Design Tools and Techniques.

General Terms
Algorithms, Design

Keywords
Pareto optimality, multi-objective fitness, software design, search-based software engineering, software architecture

1. INTRODUCTION
Software architecture design has been traditionally regarded as an art rather than as a systematic process. Despite recent efforts to systematize software architecture design, the process of software architecture design is still insufficiently understood. In particular, it would be important to explore the possibilities and limits of automated software architecture synthesis.

We have previously [7, 8] applied genetic algorithms (GA) with a simple weighted fitness function for this purpose. However, rather than producing a single architecture proposal using a weighted fitness function, a more appealing approach is to produce a palette of proposals ranging from one quality extreme to the other. This kind of multi-objective optimization can be achieved using Pareto optimality [2].

In this paper we extend previous studies in genetic synthesis of software architecture by employing Pareto optimality. To evaluate the results of our work, we have genetically synthesized a palette of architecture proposals for an example system using the Pareto approach. Then we have imitated an ATAM [1] evaluation for the resulting Pareto front architectures.

GECCO'11, July 12–16, 2011, Dublin, Ireland.
ACM 978-1-4503-0690-4/11/07.

2. RELATED WORK
Recently, approaches dealing with software design have gained interest in search-based software engineering [4, 6]. Pareto optimality has been applied surprisingly seldom in search-based software engineering, and mostly on areas irrelevant to this study. One relevant study using Pareto optimality is has been made in refactoring [5].

3. METHOD
For expressing functional requirements we identify the primary use cases of the system, and refine them into sequence diagrams. The sequence diagrams are then transformed into a class diagram (the so-called null architecture). Our example system is the control system of an electronic home (called hereafter ehome). The functional requirements for ehome lead to 56 operations and 90 dependencies between the operations, and the null architecture for ehome contains 12 classes.

The architectural data is encoded into a chromosome form. The chromosome consists of operation specific supergenes. Each supergene has a several fields for the data particles regarding one operation. The fields contain data deduced from the sequence diagrams, as well as data regarding the operation's place in the architecture. For a more thorough description on the supergene encoding, see [7].

Architectural design means here the application of various standard architectural solutions called collectively patterns. For these patterns we have chosen two architectural styles [9] (message dispatcher and client-server), and five design patterns [3] (Façade, Mediator, Strategy, Adapter and Template Method). The mutations are implemented in pairs of introducing a specific pattern or removing it. The crossover operation is implemented as a traditional one-point crossover.

We say that a solution $x^* \in F$ is Pareto optimal if for each $x \in F$, we have either $f_i(x) = f_i(x^*)$, for all i = 1, …, p, or there is at least one property i such that $f_i(x) < f_i(x^*)$. That is, x^* is Pareto optimal if there exists no feasible solution x that increases some criterion without causing a simultaneous decrease in at least one other criterion. Typically, there is not a single solution that is Pareto optimal, but a set of Pareto optimal solutions (the Pareto front) [2]. In our context, the Pareto front consists of the architectures that are Pareto optimal in the populations created by GA. Selecting the individuals for each generation is made by iterative selection of Pareto fronts. The final result consists of the Pareto front in the last generation created.

The fitness function used here measures modifiability and efficiency. It calculates the amount of calls between and within classes and inspects how patterns are used, and how much the message dispatcher and client-server architecture styles are applied. Both quality attributes should be maximized.

4. EXPERIMENTS AND EVALUATION

We used the ehome sample system to test our approach. In our experiments we used a population size of 100, 250 generations and did 20 test runs. The collected Pareto fronts are presented in Figure 1, where a clear trend can be seen. It can be seen, that efficient solutions are clustered, while modifiable solutions are much more sparse. This is natural, as most efficient solutions are those with minimal amount of mutations applied, while modifiability can be reached in different ways through application of patterns, and thus it would be unnatural to have such a clustered Pareto front on the "modifiability side" as can be seen on the "efficiency side".

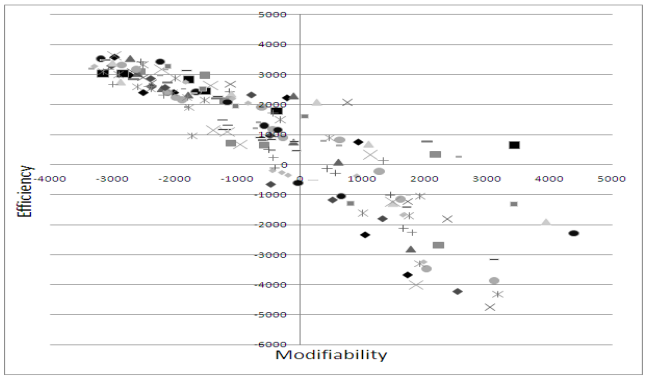

Fig. 1. Pareto fronts of 20 runs after 250 generations

We validated our results by studying whether the quality distribution of the architectures in the Pareto front corresponds to the quality distribution of the architectures with respect to an imitated ATAM evaluation [1]. To carry out this evaluation, we produced five efficiency related scenarios (where penalty should be minimized) based on use cases, and collected 12 modifiability scenarios (where reward should be maximized) from external experts.

We used the final Pareto fronts of five random runs for the validation. Each individual in the Pareto fronts was given penalty and reward points based on how it performed in the efficiency and modifiability scenarios, respectively. The points of both quality attributes were then compared against the rank the individual had in the Pareto front regarding each quality attribute. Figure 2 shows the scatter plots for scenarios. Each Pareto front has a distinct marker in the plots, and the y-value is the rank. As can be seen, the solutions with foremost ranks perform the best and those in the last ranks perform the worst in the scenarios as well, while in the middle there is some variation.

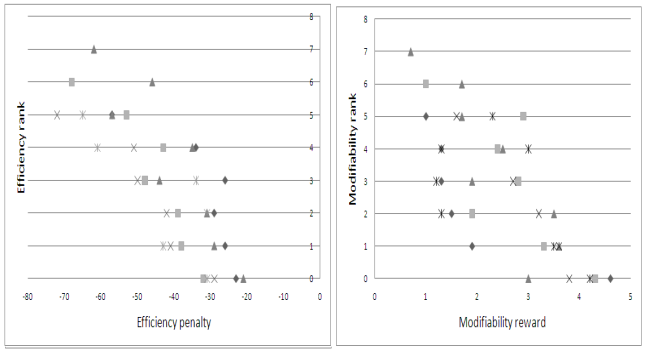

Fig. 2. Scenario plots

5. CONCLUSIONS

This work suggests that the basic problem of conflicting goals in software architecture design, critical in genetic approaches, can be solved satisfactorily using Pareto optimality. We showed that the quality distribution of the architectures in the Pareto front is similar to the distribution obtained using imitated ATAM evaluation.

Acknowledgements: The research was funded by the Academy of Finland (project Darwin).

6. REFERENCES

[1] Clements P., Kazman R., and Klein M. 2002. *Evaluating Software Architectures*. Addison-Wesley.

[2] Coello Coello, C. C. 2000. An Updated Survey of GA-Based Multiobjective Optimization Techniques. *ACM Computing Surveys* 32, 2, 109-143.

[3] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. 1995. *Design Patterns, Elements of Reusable Object-Oriented Software*. Addison-Wesley.

[4] Harman, M., Mansouri, A., and Zhang, Y. 2009. *Search Based Software Engineering: A Comprehensive Analysis and Review of Trends, Techniques and Applications.* Technical Report TR-09-03, Dept. of Computer Science, King's College London.

[5] Harman, M., and Tratt, L. 2007. Pareto optimal search based refactoring at the design level. In *Proc. GECCO'07*, 1106-1113.

[6] Räihä, O. 2010. A survey on search-based software design. *Computer Science Review* 4, 4, 203-249.

[7] Räihä, O., Koskimies, K., and Mäkinen, E. 2008. Genetic synthesis of software architecture. In *Proc. SEAL'08*, LNCS 5361, Springer, 565-574.

[8] Räihä, O., Hadaytullah, Koskimies, K., and Mäkinen, E. 2010. Synthesizing architecture from requirements: a genetic approach. *Relating Software Requirements and Architecture*, *(Ch.14)* Springer, to appear.

[9] Shaw, M., and Garlan, D. 1996. *Software Architecture – Perspectives on an Emerging Discipline*. Prentice Hall