Identifying "Good" Architectural Design Alternatives with Multi-Objective Optimization Strategies

Lars Grunske
Boeing Postdoctoral Research Fellow,
School of Information Technology and Electrical Engineering (ITEE), The University of Queensland
4072 St Lucia, Brisbane, Australia
Tel: + 61 7 33651648
grunske@itee.uq.edu.au

ABSTRACT

Architecture trade-off analysis methods are appropriate techniques to evaluate design decisions and design alternatives with respect to conflicting quality requirements. However, the identification of good design alternatives is a time consuming task, which is currently performed manually. To automate this task, this paper proposes to use evolutionary algorithms and multi-objective optimization strategies based on architecture refactorings to identify a sufficient set of design alternatives. This approach will reduce development costs and improve the quality of the final system, because an automated and systematic search will identify more and better design alternatives.

Categories and Subject Descriptors: D.2.2 [Software Engineering]: Design Tools and Techniques—Computer-aided software engineering (CASE), Evolutionary prototyping

General Terms: Design, Reliability, Economics

Keywords: Multi-Objective Optimization, Dependability, Architecture Trade-off Analysis, Architecture Refactorings,

1. INTRODUCTION

The development of software-intensive technical systems is challenging, because software and system engineers have to deal with a large number of non-functional or quality requirements such as safety, availability, reliability, maintainability and temporal correctness requirements. One major difficulty is that these non-functional requirements conflict with one another and with economic constraints. To construct a system that fulfills all its quality requirements is often not possible. As a consequence, system engineers have to consider several design alternatives and identify a solution that fulfills most quality objectives. This process is called trade-off analysis. For economical reasons this process should be applied as early as possible [4]. Consequently, the best time is in the architecture design phase, because the architectural specification is the first system model that allows the prediction of non-functional attributes. Furthermore, architectural design decisions have a large impact on the quality of the final system and bad architectural decisions cannot be fixed by corrective measures later in the development process.

To find good design alternatives that help system architects perform trade-off analysis is a cost and skill intensive task. The basic intention of this task is to find a set of architecture specifications that solve a multi-objective optimization problem, where the objectives represent different quality attributes. This paper introduces a software engineering process based on evolutionary algorithms, where an architecture specification that fulfills its functional requirements is taken as an initial solution. Based on this initial solution, "child"-architectures are generated by applying quality-improving, behavior-preserving architecture refactorings. Examples of architecture refactorings include: triple modular redundancy (TMR); dual channel redundancy; watchdog; and recovery blocks. The selection of good architectures, which are used for reproduction in the next iteration of the evolutionary algorithm, is based on the results of architecture evaluations, which should identify the grade of fulfillment of non-functional requirements.

The rest of this paper is organized as follows. Section 2 gives an overview of current quality optimization approaches and describes related work in the area of architecture evaluation and transformation. Section 3 introduces the idea of using multi-objective optimization strategies and evolutionary algorithms for identifying good decisions in the architecture design phase. This idea will help to improve trade-off analysis techniques. Some initial results obtained from a satellite control system case study are presented in Section 4. Finally, Section 5 concludes the paper with a discussion, summary and directions for future research.

2. CURRENT APPROACHES

For the optimization of non-functional properties current approaches [2] use the cyclic process presented in Fig. 1. The precondition for the process application is an architecture specification that fulfills all functional requirements. Based on this specification the non-functional properties are determined by architecture evaluations. If the architectural specification does not meet its nonfunctional requirements, the software architecture must be restructured (e.g., with quality-improving architectural transformations). These transformations should influence the non-functional properties without changing the functional behavior. Thus, after the transformation, the architectural specification is still correct. If all non-functional properties meet their requirements, the process can be terminated and the system development can be continued with the detailed design.
The benefits of this process are its simplicity and universality. The process can be tailored to different non-functional properties by using specific evaluation methods and architecture transformations. Overviews of useful evaluation and transformation techniques are given below in section 2.1 and 2.2.

One weakness of this process is its brute force strategy, which will result in a huge computational effort for complex architecture specifications and quality landscapes. Additionally, this process focuses on only one quality attribute at a time. As a consequence, such approaches are prone to falling into local optima for multi-objective quality optimization problems.

2.1 Architecture Evaluation

For the evaluation of software architectures, many specific approaches for the different non-functional properties are described in the literature [1, 4, 14, 10]. For the cyclic optimization process described in Fig. 1, quantitative approaches are relevant. These quantitative approaches determine measurable characteristics that can be used to compare different architecture options and provide an indication of whether a system will fulfill its quality requirements.

For different non-functional properties, different quantitative evaluation techniques and models have been proposed. As an example for the safety evaluation Component Fault Trees [15], Failure Propagation and Transformation Notation-Modules [8] and State Event Fault Trees [13] can be used. Performance attributes can be evaluated at an architectural level, with performance evaluation models, such as [Layered/Extended] Queueing Networks, [Stochastic/ Timed] Petri Nets and Markov Models. For a detailed description of these performance models we refer to the survey in [1]. A comparison of several reliability evaluation techniques is presented in [10]. The research to predict real-time properties is mostly concerned with analyzing the schedulability of architectural specifications and whether or not a system can meet hard deadlines. Consequently, specific scheduling models (RMA, EDF, etc.) have to be considered and it must be determined if all architectural components meet their deadlines. If all components fulfill their deadlines, an end-to-end analysis could be performed [7], which identifies the worst-case reaction time of a system to handle a request from the environment. To estimate development costs or life cycle costs at the architecture design phase many of the traditional techniques (e.g., COCOMO II) can be used. These cost estimation techniques are based on design metrics, and cost prediction models, which include several project-specific factors. A cost prediction method, which also includes the value of non-functional properties, is the Cost Benefit Analysis Method (CBAM) [4]. Together with the traditional approaches CBAM could be used as a convenient cost prediction method.

2.2 Architecture Transformation

Architecture transformations that preserve the system’s behaviour and increase quality attributes of the architecture are used in many engineering disciplines. For software engineering these behaviour-preserving architecture transformations are similar to refactorings at the code level and thus they are called architectural refactorings.

The research on architectural refactorings follows basically two streams. The first stream aims to automate the application of refactorings. Consequently, refactorings must be formally specified and a mechanism must be provided to automatically apply them. Since most architecture specifications are defined as graph-based structures, graph transformations are identified as a suitable formalism to formalize refactorings [12]. The second aims to create libraries or pattern collections that describe a group of suitable refactorings to improve one non-functional property (e.g., reliability [16], safety [11] and realtime refactorings [7]).

3. MULTIOBJECTIVE OPTIMIZATION STRATEGIES

Many real-world optimization problems have multiple and conflicting objectives. As a result, one single solution for these problems often doesn’t exist and traditional optimization techniques are not suitable. To deal with multi-objective optimization problems several techniques have been developed [9]. The aim of these techniques is to find a set of solutions, where none of the solutions of this set is superior to the others. These solutions are known as Pareto-optimal solutions [6, 9, 5].

3.1 Formal Problem Definition

As specified in [5] a multi-objective optimization problem can be defined as follows:

\[
\text{Find a solution } x, \text{ which is an element of the solution space } X, \text{ satisfies a set of constraints } g_i(x) \text{ and optimizes a vector function } f(x) = \{f_1(x), f_2(x), \ldots, f_n(x)\}, \text{ whose elements represent the objective functions.}
\]

If this definition is applied to the quality optimization problem in the architecture design phase, the solution space \(X\) describes all possible architecture specifications that fulfill the functional requirements. A constraint \(g_i(x)\) could describe budget limitations, technical feasibility constraints or a certain threshold for non-functional properties (e.g., tolerable hazard probabilities). A function \(f_i(x)\) of \(f(x)\) evaluates a specific quality (e.g., reliability, timelines) of the architecture \(x\).

3.2 Evolutionary Algorithms

Evolutionary algorithms are preferred for solving multi-objective optimization problems [5, 6]. These evolutionary algorithms follow the idea of natural evolution and try to optimize or improve a population of candidate solutions, in contrast to the single solution, which is used in traditional optimization strategies. Generally, evolutionary algorithms can be represented by an iterative procedure that contains the following steps:

---

Figure 1: Cyclic process to optimize non-functional properties in the architecture design phase
• Step 1: Generate an initial population and rank the members by their fitness
• Step 2: Select a subset of the population (parents) for reproduction (based on their ranking)
• Step 3: Generate a set of child solutions by applying genetic operators (reproduction, crossover, mutation etc.)
• Step 4: Add the child solutions to the population
• Step 5: Rank the members of the population and select a diverse subset of the population for the next iteration
• Step 6: If no termination criterion is reached go back to step 2

To deal with multi-objective optimization problems this generic procedure has to be adapted in the following ways. Firstly, the ranking of individuals in step 2 and 5 must include all objectives. One simple, but limited, approach is to use a weighted sum of the objective functions, where the weights represent customer preferences. Another approach uses the concept of dominating solutions, where a solution \( x_1 \) is dominated by another solution \( x_2 \), if \( x_2 \) matches or exceeds \( x_1 \) in all objectives. Based on this concept, an integer value representing the number of dominating solutions can be assigned to each individual and the individuals with the lowest number are ranked best. Other ranking approaches can be found in the literature [9]. Secondly, diverse individuals should be selected for the next iteration in step 5, since we are interested in a broad set of solutions. This will result in a better chance to find a large set of Pareto-optimal solutions and help to avoid local optima.

3.3 Application to the Trade-Off Problem

If we tailor this generic procedure to the quality optimization problem in the architecture design phase, the initial population must contain architecture specifications that fulfill all functional requirements. The ranking procedure is based on the results of the architecture evaluations, which should identify the grade of fulfillment of non-functional requirements (as described in Section 2.1). Furthermore, architecture refactorings are used as mutation operators, which mean that child architectures are generated by applying one or more architecture refactorings to parent architectures. If all architecture refactorings are behavior-preserving and the initial architectures of a population fulfill the non-functional requirements, then each element of the population in each generation fulfills the functional requirements. As a result, the non-functional properties of the architecture specification could be improved without considering about functional properties.

Since evolutionary algorithms can only find a set of Pareto-optimal solutions, where none of the solutions could be identified as best by means of the multiple objectives, the final decision (the selection of the used architecture for the detailed design) must be taken by the architect. However, since the search space of architecture decision problems are often overwhelming; the benefit of using evolutionary algorithms as an input for decision making of an architecture trade-off problem result from the automated and systematic search, which will identify more and better design alternatives.

4. INITIAL RESULTS

To test the evolutionary algorithm presented in Section 3, a satellite control system based on the BIRD (Bi-spectral InfracRed Detector) satellite specification [3] is used as a case study. The mission of this satellite is to detect high temperatures on the earth surface, such as wildfire, with its infrared cameras. To keep the example simple only two high-level functions are investigated. The first function is the Attitude Control Function (ACF) intended to control the satellite’s position and rotation. The second function, WildFire Detection (WFD), collects infrared sensor data and transmits the data to the ground station. The reliability of both functions is crucial for the mission’s success, because once the satellite is in the orbit it cannot be repaired. Consequently, the architecture of the BIRD Satellite uses effective redundancy and fault detection mechanisms.

To prove the applicability of evolutionary algorithms for the multi-objective optimization problem, we use a simplified architecture (see Fig. 2) without any redundancy. Additionally, to avoid common cause failures the Power Supply -unit is only used by the Board Computer.

![Figure 2: Simplified Architecture of the Satellite](image-url)

In this architecture specification, quality attributes are annotated to each component. These quality attributes include cost and weight estimations, as well as constant reliability probabilities. Based on these attributes the cost and weight of the system can be determined by a summation of the component costs and weights, if we assume no integration overhead. The reliability of the system functions can be determined with Reliability Block Diagrams (RBDs), where the RBD for the first function is a series configuration of the components: Board Computer, Power Supply, Sun Detector, Gyroscope, Star Sensor and Reaction Wheel. The RBD for the second function is defined by a series configuration of the components: Board Computer, Power Supply, Sun Detector, Infrared Camera and EarthLink Transmission System. As a mutation operator, we use the two-channel redundancy pattern [7]. The objectives are (a) keep the cost of the satellite as low as possible and (b) improve the reliability of the two major satellite functions. Finally, since the satellite must be launched as a small satellite the upper limit of the weight is 100 kg. This limits the number of redundant components, which can be used in the satellite.

Based on this problem, our prototype implementation of the evolutionary algorithm identifies the solutions repre-
sent in Fig. 3, which are Pareto-optimal (compared to the global solution space), for the trade-offs between the reliability of function one and two and the satellite’s cost. The coding of the solutions must be interpreted as follows: each element in the string represents a component according to the identifier in Fig. 2 and the number identifies how many redundant components are used. As an example in the architecture (2,1,1,2,1,1,1,1) the components Board Computer and Star Sensor are implemented redundantly.

![Figure 3: Pareto-optimal solutions obtained by applying an evolutionary algorithm](image)

5. CONCLUSION & FUTURE RESEARCH

The initial experiment and the prototype implementation have shown that evolutionary algorithms and multi-objective optimization strategies can be used to identify Pareto-optimal or near Pareto-optimal architecture designs, which can be used as an input for architecture trade-off analysis techniques. The main benefit of this approach is the ability to handle the large search spaces of quality optimization problems and to automate the identification of good architecture design alternatives. This will reduce development costs and improve the quality of the final system. Additionally this approach enables the ranking of design decisions (architectural refactorings) with respect to conflicting quality requirements.

The presented results based on the BIRD satellite case study are admittedly of a preliminary nature and the prototype implementation of the evolutionary algorithm is limited. In particular, the quality evaluation methods used to rank the different design alternatives are over-simplified (RBDs) and do not consider quality dependencies between different components. To get better results the state of the art architecture evaluation techniques mentioned in Section 2.1 will be implemented in the future. As a mutation operator only the two-channel redundancy pattern is used. To create better design alternatives other architecture refactorings must be used and the selection of appropriate refactorings should be guided for general problem scenarios. Finally, compared to real-world systems, the BIRD-satellite is small and has a very limited problem/solution space. This has the benefit that the results can be validated by calculating and interpreting the results manually. However, it remains to be proven that the approach can handle complex and convex solution spaces in an acceptable time with an acceptable diversity of solutions.

6. REFERENCES