A Prediction Approach to Support Alternative Design Decision for Component-Based System Development

ADIL A.A SAED, WAN M.N. WAN KADIR, ADIL YOUSIF
Department of Software Engineering
University Technology Malaysia
Skudai, Johor 81310 UTM,
MALAYSIA
Aasadil2@live.utm.my, Wnasir@cs.utm.my, adiluofk@gmail.com

Abstract: - Interpreting the results of performance analysis and generate alternative design to build component system is a main challenge in the software performance domain. Improving one quality feature can weaken another; quality features cannot be individually improved. Furthermore, the span of design space hinders the selection of the appropriate design alternative. In the context of Component-based system, the paper discusses the assessment of performance characteristics of software architecture, auto-generation of the new candidates, as well as relevant concepts to optimization problems such as design space and degree of freedoms. We introduce an approach supports the alternative design decision using PSO as a promised meta-heuristic technique. Performance cannot be assisted in isolation of other non-functional properties; we outline the process of evaluating the software performance considering the probability of its conflicting with reliability. Therefore, the proposed approach enables the architect to reason on the auto-provided architectures and chooses the optimal solution. Consequently, better architecture design could be obtained and time to develop the system will be reduced. Finally, a simple case study is illustrated in the paper as an example to demonstrate the applicability of the approach.

Key word-s: Metaheuristic, Performance, Component-based system, Design space, PSO

1. Introduction
Component-based System Development (CBSD) enables software architects to reason on the composed structure. This is not only essential for the functional properties but also non-functional properties and software quality as well. Performance, which is referring to how extend the system or component has satisfied the predefined requirements on restrictions of specific factors such as accuracy, available memory usage [1], is an important non-functional characteristic that must be deliberated when developing such applications. In the same time, it is an essential attribute for software quality. The problems in software performance often leads to several difficulties such as; financial loss, costly development, and further than that, the damage of relationship with clients. When performance issues are addressed earlier throughout development, correction of problems will positively impact on the cost, schedule, and quality of the software [2].

Software architects are employing Model-Driven Development (MDD) [3] to manage architectural models of the system under development. The models are transformed into simulation-based or analytical models (e.g. stochastic process Algebra, QN, and Petri nets). From the resulted model, the performance metrics such as response-time, throughput, and resource utilization can be calculated. This resulted analysis is still lacked of automation and is based on the skills and experience of analysts [4]. Although many approaches have been proposed and were successfully applied to predict software performance, the span of design space is a substantial problem when attempting to select the optimal design. Due to the difficulty and complexity of manually selecting the best combination, meta-heuristic approaches have been used to deal with this problem.

Meta-heuristics are originated and inspired by natural process and creature’s to solve complex real world problems. Optimization is at the heart of many natural processes such as; Darwinian evolution, social group behavior and foraging strategies. The last two decades have witnessed notable increasing in the domain of nature-inspired search and optimization algorithms. Recently, these techniques are applied to variant problems. Evolutionary computing methods and the swarm intelligence algorithms are the main groups represent the field[5].

Meta-heuristics evolutionary techniques such as Genetic Algorithms (GAs) methods have proven its
usefulness to solve the problem of spanned design space. In recent researches Swarm Intelligent (SI) techniques such as Particle Swarm Optimization (PSO) [6], [7], [8] an alternative search technique, often performed better than GA when applied to various problems [9, 10]. Evolutionary techniques need to handle the population movement; therefore, they are less fast in discovering optimal solutions. Furthermore, evolutionary algorithms may have a memory to store previous status; this may help in minimizing the number of individuals close to positions in candidate solutions that have been visited before, but it may also slow to converge since successive generations may die out. In recent researches Swarm Intelligent (SI) techniques such as PSO has been applied in software testing namely unit testing [21]. However, the generation of appropriate test cases for a software unit impacts the quality of the overall testing of this unit. GAs was the common search algorithms to find such cases. In this paper, we attempt to answer the question of how PSO, as an alternative algorithm, contribute to the software testing in context of evolutionary structural testing. The researcher concludes that; GA features a slightly faster convergence for simple functions whereas PSO outperforms GA primarily for complex functions with big search spaces.

Here we introduce a novel approach that search the design space and automatically generate alternative's design. The approach enables architects to reason on the provided options and chooses the best solution that satisfies performance requirements.

2. Related Work

There are two main approaches deal with the problem of interpreting the performance result; rule-based and metaheuristic. Rule-based [12] approaches try to identify problems in the model (e.g. bottlenecks) based on predefined rules and rules containing performance knowledge are applied to the detected problems. Rule-based approaches focus on performance analysis without considering other quality criteria. They operate on the performance model instead of the architectural model and are therefore difficult to use for regular architects not familiar with performance formalisms. These approaches cannot find solutions for which no rule exists, thus, they cannot cover all possible solutions and might result in locally optimal solutions. Cortellessa and Frittella [11] have proposed an automated approach for the performance feedback generation process based on performance anti-patterns. By applying the approach in simple case study, they describe how anti-patterns are detected in an XML format. In this method, the detection of antipatterns in a subsystem is a task whose complexity heavily depends on the structure of the subsystem and the definition of the antipatterns itself. Furthermore, there is no offer of new architecture candidates.

Metaheuristic-based approaches (e.g., [14] [15] [16], [17]) encode the challenge of improving architectures as an optimization problem and use metaheuristic search techniques [18] [19] (e.g., genetic algorithms, simulated annealing, etc) to find better design models. Noorshams et al [20] have extended the existing Quality of service Modeling Language to enable the specification of optimization goals and quality requirements. GA is used as optimization technique and the case study in PCM. The approach is base on identifying forbidden space, so the search focuses on specific design space. Our approach is based on this method, but we claim that, by using PSO after translation the QML requirements to constraints in an optimization problem, our approach will be produce better efficient results especially with large design space. Therefore the software architect can effectively and efficiently better solution.

PSO has been applied in software testing namely unit testing [21]. However, the generation of appropriate test cases for a software unit impacts the quality of the overall testing of this unit. GAs was the common search algorithms to find such cases. In this paper, we attempt to answer the question of how PSO, as an alternative algorithm, contribute to the software testing in context of evolutionary structural testing. The researcher concludes that; GA features a slightly faster convergence for simple functions whereas PSO outperforms GA primarily for complex functions with big search spaces.

Here we introduce a novel approach that search the design space and automatically generate alternative's design. It enables architects to reason on the provided options and chooses the best solution. Consequently, better architecture design could be obtained and time to develop system could be reduced.

3. Problem Formulation

In this section we describe how the problem originates in the context of developing system base on components. Then, the formulation of problem as optimization challenge is illustrated. Finally, as the PSO is the nominated technique to be applied in this research, general algorithm has been demonstrated.

3.1 Problem in development of Component-Based System

When an architect starts building a new CBS application, he has many options to do the job. Each probable solution is arranging from a mixture of distinctive components. All those possible alternatives are called Design Options. The
combination that satisfied the performance requirements is the target of the architect. However, design options are proportional with the degree of freedom. The degrees of freedom are resulted due to the following possible changes [22];

**Components selection:** refers to the selection of one component from number of components with the same functionality but different performance specifications.

**Resource Allocation:** due to the fact that, the selection of hardware does not impact the functional of components, its configuration could be changed during search. Therefore, hardware environment is modeled separately from the common assembly. This will enable the adjusting of resource size.

**Setting Parameters:** performance of whole system may affected by parameters assigned by software component. Those parameters, often, do not have functional impact. The option of using RMI or SOAP network protocols can be configured.

**Usage Profile:** different options related to number of users and their inputs could impact performance. If those variables are represented as parameters, trades-off between scalability and design options will be available.

Therefore, the design-space composes of various, but countable number of candidates. Obviously, those candidates are directly proportional with the degree of freedoms. In many cases, the design space become too large, this might impact on the task of search for better solution. As stated before, meta-heuristics such as Gas techniques have been widely used to solve the problem of spanned design.

### 3.2 General optimization problem formulation

Assume general optimization problems \( P (S, f) \) which can be identified as follow:

a) an objective function \( f \) to be minimized or maximized, the minimization problem is taken as general, where

\[
f : D \times D \rightarrow IR
\]

b) a set of variable \( X \)

\[
X = \{x_1, \ldots, x_n\};
\]

c) variable domain

\[
D_1, \ldots, D_n;
\]

d) constraints and how deal with the velocity (according to specific strategy; delete, edit, modify, or penalty)

The set of probable solutions with condition of satisfying all the constraints denoted as:

\[
S = \{ s = (x_1, v_1), \ldots, (x_n, v_n) \mid v_i \in D_i, s \}
\]

(Under the constraints, where \( S \) the area of the search, Design Space.)

The solution with minimum objective function value is called global optimal solution. This solution is linked to what known as; set of global optimal solution.

Assume the globally solution for \( P (S, f) \) is;

Satisfy

\[
f(x^*) \leq f(s) \forall s \in S
\]

where \( S^* \subseteq S \) is the set of Global Optimal Solution

### 4. Problem Solution

The general PSO algorithm has been presented in this section, followed by the proposed approach, and finally the example to apply is demonstrated.

#### 4.1 Particle swarm optimization

Particle Swarm Optimization (PSO) is one of the swarm intelligent (SI) optimization methods, it’s an adaptive optimization method [23, 24] inspired by social behavior of swarms such as bird flocking or fish schooling. In PSO, particles never die, and particles are considered as simple agents that move and interact through the search space and record the best solution that they have visited. Each particle represents a candidate solution in the solution search space and each particle has a position vector and velocity. To update the particle’s positions the following equations are used:

\[
V[i] = v[i] + c1 \cdot \text{rand()} \cdot (pbest[i] - present[i]) + c2 \cdot \text{rand()} \cdot (gbest[i] - present[i]) \quad (1)
\]

\[
\text{present}[i] = \text{persent}[i] + v[i] \quad (2)
\]

Where \( c1 \) and \( c2 \) are learning factors (weights)
The behavior of the particles is subjected to their capability to train from their past personal experience and from the success of their neighbor to adapt the flying speed and direction to the target. However, particles manage the current position, velocity and personal best position. Beside the personal best solution, the swarm is targeting the global best solution (See Figure 1) which is followed by pseudo algorithm.

As it can be seen from the flow chart in Figure 1, the search is guided by the fitness function with stop criteria for obtaining optimal or sub-optimal solutions applying the following steps:

- Initialize the initial population with randomly generated particles (Each particle represent a feasible solution in search space).
- Update through generations in search for best solution
- Each particle has a velocity and position
- Update for each particle uses Pbest and Gbest.
  - Pbest: best solution (fitness) it has achieved so far. (The fitness value is also stored.)
  - Gbest: best value, obtained so far by any particle in the population.
- pBest is the best local particle that satisfy the fitness function.
- gBest is the best global particle that satisfy the fitness function.

![Figure 1: Process Flow of General Particle Swarm](image)

**4.2 Solution processes**

4.2 Solution processes

Palladio Component Model (PCM) [25] and the PSO have been used to describe the application of the method. PCM has been chosen to enable the prediction of selected quality criteria’s automatically, where Performance and reliability are provided and other quality criteria might be extended as well.

Figure 2 illustrates the process model of our method. The process begins with an initial model of component-based software architecture (initial candidate) and modifies it along the given degrees of freedom. Following are some details of the steps:

- Analysis design options; in this step the available design options are analyzed in the given software model based on the given degrees of freedom.
- Search settings; the problem is translated into a general form that can be searched by metaheuristic techniques. Metaheuristic search techniques search in several iterations until a stop criterion is fulfilled. In each iteration, a population of candidates is generated, next, fitness function evaluated and then new population extracted. The search can be initiated with random population. In this work we prefer to starts with a population specified by the user.
- Run Search: the search is running until the stop criteria met.
- Finally, the resulting set of Pareto-optimal solutions is presented to the software architect. The software architect can identify interesting solutions and make well-informed trade-off decisions between the multiple quality criteria.

**4.3 Application on an example**

The case study of Business Report System (BRS) is selected to evaluate the approach. The BRS provides statistical reports about business processes and is loosely based on a real system. The system composed of nine components and is allocated to four servers as explained below:

One usage scenario, user interacts with the system every five seconds requesting a sequence of reports and views. For generating reports or viewing the plain data logged by the system, the web Server component deals with user requests. It delegates the requests to a Scheduler component, which in turn forwards the requests to the ReportingEngine component. The ReportingEngine accesses the...
Database component, sometimes using an intermediate cache component.

Figure 2: Overview of the proposed Approach from viewpoint of search technique

Each component can be allocated to one of the four servers with varying configuration. We realized the server configuration and allocation with joint degrees of freedom. Overall, 16 combinations were available for the four base servers $S_i$ and four configuration options $C_j$: $S_1C_1, S_1C_2, C_1S_3 \ldots S_4C_3, S_4C_4$. The Web server can be realized using third party components. The software architect can choose among three functional equivalent implementations: Webserver2 with cost 4 and Webserver3 with cost 6. Both have less resource demand than the initial Web server. Webserver2 has better availability for the requests of type “view,” while Webserver3 has better availability for the requests of type “report.”

Table 1: Sample of results using GA [20]

<table>
<thead>
<tr>
<th>No</th>
<th>Candidates</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(10.01, 21.64, 44.93, 5.37, WS2, Ser1, Ser2, Ser4, Ser3)</td>
</tr>
<tr>
<td>2</td>
<td>(31.86, 23.68, 51.20, 9.88, WS, Ser1, Ser3, Ser3, Ser3, Ser1)</td>
</tr>
<tr>
<td>3</td>
<td>(12.52, 33.66, 37.92, 12.37, WS, Ser4, Ser2, Ser3, Ser3, Ser3, Ser2)</td>
</tr>
<tr>
<td>4</td>
<td>(12.52, 33.66, 37.92, 12.37, WS, Ser2, Ser2, Ser3, Ser4, Ser2)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>No</th>
<th>Candidates</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>(29.48, 31.26, 43.23, 8.0, WS1, Ser3, Ser3, Ser2, Ser4)</td>
</tr>
<tr>
<td>26</td>
<td>(10.01, 21.57, 38.20, 11.60, WS4, Ser3, Ser4, Ser2, Ser3)</td>
</tr>
</tbody>
</table>

Table 2: Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of particles</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inertia weight $\omega$ Lin</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Acceleration coefficient $c$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Velocity bound $V_{\text{max}}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Refreshing gap $m$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Boundary condition</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Termination</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Finally, the software architect makes the trade-off decision and chooses one of the solutions.

5. Conclusion

Designing architectures that provide a good trade-off between multiple quality criteria is quite difficult. Architect needs to map the result back in the design model and then manually develop new alternative. Most of the proposed approaches just
recognize that performance is not satisfied. Optimization techniques such as Meta-heuristic approaches can be employed to search and generate design alternative.

This paper has identified and delineated relevant concepts to optimization such as design space, metaheuristic algorithms, and degree of freedom. Since performance cannot be assisted in isolation of other non-functional properties, we outlined the process of evaluating software performance considering the probability of its conflicting with reliability. We proposed the use of multi-objective optimization rather than single objective in the context of component-based development.

PSO is similar in some ways to genetic algorithms and other evolutionary algorithms. However, PSO requires less computational book keeping, and generally only a few lines of code and fewer parameters are needed. Besides, it is easy to understand and implement. Therefore, the paper proposed an approach uses the PSO in the field of performance prediction. The approach contributed to automate and explore of design space, so architects could be able to choose better alternative architecture model. The improvement of architecture encompasses faster and more reliable software components, a more powerful hardware environment, or a different allocation of components to available hardware. Finally, a simple example was depicted in the paper to demonstrate the applicability of the approach. Our objective to the next step is to apply our method to a real case study, and then comparing its results to results of other approaches in the literature.

Acknowledgement
The authors would like to thank Universiti Teknologi Malaysia for their financial support under the Research University Grant Scheme (RUGS) Vot No. Q.J130000.7128.01H13.

References


