The handle http://hdl.handle.net/1887/30105 holds various files of this Leiden University dissertation.

**Author:** Etemadi Idgahi (Etemaadi), Ramin  
**Title:** Quality-driven multi-objective optimization of software architecture design: method, tool, and application  
**Issue Date:** 2014-12-11
Chapter 6

New Degrees of Freedom for Software Architecture Optimization

This chapter addresses RQ2 which is mentioned in Section 1.2:

Can enlargement of the optimization search space help the meta-heuristic approach to find better architectural solutions?

We know the component-based paradigm makes it possible to easily and automatically create variation of architectural designs. Hence, the component-based paradigm is key to meta-heuristic optimization approaches, such as genetic algorithms (GA), to automatically generate new alternative solutions. However, to guarantee that the variation process does not change the functionality of the system, these approaches should only consider variations of architectural designs that do not modify the interfaces of components. The ways in which a candidate solution architecture can be varied without changing the functionality are called Degrees of Freedom.

This chapter presents two novel degrees of freedom for the optimization of software architectures. These two degrees of freedom are: (i) the topology of the hardware platform, and (ii) the replication of software components. The results show us that they can improve the results of the optimization algorithm by enlarging design space. This chapter analyses these two new degrees of freedom by running a very computationally-intensive experiment of an industrial case study.

This chapter is structured as follows. First, details of two new proposed degrees of freedom (DoFs) are discussed in Sections 6.1 and 6.2. After that, with two experiments based on two different systems, the effects of the proposed degrees of freedom for finding better solutions and enlarging design space have been analysed. These experiments and their results are presented in Sections 6.3, 6.4. Section 6.3 shows an
experiment based on Cruise Control case study (see Section 5.2) which has been run with only topology DoF. Section 6.4 demonstrates a larger experiment which is based on SAAB Instrument Cluster case study (see Section 5.3) and has been conducted with both of the new proposed DoFs. Lastly, Section 6.5 summarizes this chapter.

6.1 New Degree of Freedom 1: Topology

We start with an example in order to provide a better describe for the topology-DoF: The case study of [MKBR10] demonstrates the state-of-the-art optimization based on an initial architecture. Figure 6.1 represents the base topology used in their study. In their approach, new alternative solutions can only be created by leaving out nodes and/or buses from this base topology. Hence the manner in which nodes can be connected is ‘hard-wired’ into the initial base topology. Hence in this setting, the optimizer can achieve architectures with topologies shown in Figures 6.2a, 6.2c and 6.2e. For example, to achieve topology that depicted in Figure 6.2a the optimizer should keep Node1, Node2, Node3, Node5, Node7 and Bus1, Bus2, Bus3 and should remove Node4, Node6, Node8 and Bus4. Similar to this, to achieve Figure 6.2c the optimizer keeps Node2, Node3, Node5, Node7, Node8 and Bus2, Bus3, Bus4 and removes Node1, Node4, Node6 and Bus1.

The topologies shown in figures 6.2b, 6.2d and 6.2f can not be created from the base topology. For example, in Figure 6.2d if Bus2 is used, it must be connected to all of predefined nodes – as is the case in the base topology. In the existing optimization approaches a bus is either in or out of the architecture. Using our new DoF, we can create new topologies that contain new nodes and new connections that were not yet in the original base topology. In theory this makes the search space infinitely large. However, we use use pragmatic bounds on the size of the solution. These bounds still allow more variation of solutions that the state-of-the-art, yet limit number of computations needed by the optimizer within practical scope.

![Figure 6.1: Base topology](image-url)
New Degree of Freedom 1: Topology

(a) Possible topology 1

(b) Impossible topology 1

(c) Possible topology 2

(d) Impossible topology 2

(e) Possible topology 3

(f) Impossible topology 3

Figure 6.2: Samples of possible and impossible topologies
6.2 New Degree of Freedom 2: Replication of software components

We use an example in order to describe the replication-DoF: Assume an event in a simple system is being called every 50 ms and its deadline is 100 ms. The execution time of the related component on the most powerful processor from repository is 80 ms. Because the execution time is larger than the trigger time, the requests miss their deadlines. In this kind of situations, the replication-DoF opens the possibility to solutions where the processing is distributed across multiple processor nodes running in parallel. This allows for a common architectural solution called 'load balancing'. By applying the load balancing technique, the system can handle multiple request in parallel and thus increase the throughput of the system.

![Architecture without load balancing](image1)

![Architecture with load balancing DoF](image2)

**Figure 6.3:** Comparing the effect of load balancing DoF

These quality properties could be critical design objectives in business- or safety-critical systems. Because without having such degree of freedom in generating new solutions and its related mechanism in performance evaluation, the optimizer generates solutions which possibly miss some deadlines depends on defined scenarios and load of the system. For example, in the aforementioned system, if the architecture contains two instances of one software component, then it can respond to all requests in a way which meets all deadlines. Figure 6.3 shows the topology that corresponds with the load balancing pattern in this example.

Because the load balancing technique distributes the load of the system, it affects the architecture evaluation scores for performance, reliability and utilization. However, it should be noted that the replication pattern is different from the Voting mechanism as described in e.g. [LSBB03]. Hence, it can not improve the safety aspect. When a node in the architecture fails, the system will miss all the assigned requests to that specific node. So, the other duplicated components do not cover for that failed node.
6.3 Experiment 1: Cruise Control System

The cruise control (CC) system is a safety critical system from the automotive industry. Details of this system are described in Section 5.2. Our version of it has been derived from the VehicleControl system from [GH11], which is based on a system in [HF07]. The main purpose of the cruise control is maintaining a predetermined constant speed which is set by the driver. The software architecture of the CC system consists of six software components: IN_CONTROL, DESIRE_SPEED, AIRBAG, SECURITY_BELT, THROTTLE, and BRAKE.

The goal of this experiment is to examine the usefulness of our new topology degree of freedom. Hence, the experiment was defined in a way to generate a comparison between optimization with and without the topology DoF. Moreover, since this system is rather small, it is easy to depict the achieved optimal architectural solutions and thereby see its coverage of various topologies in the design space.

![Baseline topology for CC system](image)

**Figure 6.4:** Baseline topology for CC system

6.3.1 Optimization Setup

For solving the CC system design problem, a repository which consists of the following hardware components, has been considered:

- 28 Processors types: ranging over 14 various processing speeds from 66MHz to 500MHz. Each of these has two levels of energy consumption. A processor is more expensive if it has a smaller chance of failure.
• 15 Buses: ranging over five types of bandwidths (128, 160, 192, 224 and 256 kbps) and each type has three variants with different latencies (1, 3 and 5 ms). Again, a bus is more expensive if it supports higher bandwidth or smaller time delay.

The AQOSA optimization was run with the following parameter settings: initial population size ($\alpha$) = 100, parent population size ($\mu$) = 50, number of offspring ($\lambda$) = 50, archive size (with crowding type) = 50, number of generations = 10000, crossover rate is set to 0.95. We ran this experiment 50 times in total (25 times for both with- and without topology DoF). In the optimization without the topology DoF we used the baseline topology depicted in Figure 6.4. On the other hand, in the optimization with the topology DoF, the approach generates new topologies by itself and no baseline was used. This alleviates the architects from creating a baseline architecture.

![Graph showing response time and CPU utilization comparison](image)

(b) Best CPU Utilization

Figure 6.5: Boxplot comparison of optimization results with and without topology DoF
6.3.2 Experiment Results

Figure 6.5 shows the difference between optimization with and without the topology DoF. It shows the boxplot chart of 25 runs with topology DoF and 25 runs without it, for response time and processor utilization objectives. In both graphs, lower values for objectives indicate better solutions. Figure 6.5a demonstrates that the with-topology approach converges to solutions that are approximately 35% better for response time. Also Figure 6.5b demonstrates that optimizing with topology DoF finds solutions with on average a 12% better system processor utilization.

![Figure 6.6: Pareto front of Processor utilization vs. Cost](image)

Because the optimization of this case study considers five objectives, there are 10 2D Pareto fronts for different pairs of objectives. Figure 6.6 shows one of these fronts for the objectives Processor utilization and Cost. Each point in the plot depicts an architectural solution. The 2D Pareto fronts show the trade-off between objectives. Improving one dimension often implies a decrease of the other dimension. Different colors in the plots correspond to different numbers of nodes in the architecture: black for 1-node, orange for 2-node, magenta for 3-node, blue for 4-node, green for 5-node and red for 6-node architectures. As can be seen there is no black node which means AQOSA could not find an optimal solution with a 1-node architecture.
Moreover, to demonstrate that the approach can generate various types of topologies, one 5-node architecture (Figure 6.7), one 4-node architecture (Figure 6.8), one 3-node architecture (Figure 6.9) and one 2-node architecture (Figure 6.10) out of 50 optimized solutions have been chosen. These are highlighted in Figure 6.6 with the filled squares as well.

**Figure 6.7: Arch. A: Sample of 5-node topology**

**Figure 6.8: Arch. B: Sample of 4-node topology**
6.4 Experiment 2: The SAAB Instrument Cluster System

This experiment is conducted based on the real world industrial system of the SAAB Instrument Cluster sub-system. The system is described in Section 5.3. It consists of 18 components as depicted in Figure 5.8. The purpose of the Instrument Cluster sub-system is to provide the driver with information that is required when driving the car. The information is processed by user functions based on sensor values and presented on gauges or displays located in front of the driver.

The goal of this computationally-intensive experiment is to examine the usefulness
of both of the two new proposed degrees of freedom within the software architecture optimization process. Therefore, for this reason two comparison were targeted for this experiment: (i) comparing of best of achieved values for different objective functions, (ii) comparing the hypervolume indicators of the entire sets of optimal solutions.

6.4.1 Optimization Setup

For solving the Instrument Cluster system design problem, a repository that can use the following hardware components has been considered:

- 22 types of Processor nodes: ranging over 14 various processing speeds from 66MHz to 400MHz. Each of these has two levels of probability of failure. A processor is more expensive if it has a smaller chance of failure. (See lines 368 to 389 in Listing A.1)

- 4 types of Bus nodes: ranging over bandwidth (10, 33, 125, and 500 kbps) and each type has different latency. Again, a bus is more expensive if it supports higher bandwidth and less time delay. (See lines 390 to 393 in Listing A.1)

After defining the above hardware options, AQOSA was run with the following parameter settings: initial population size (\(\alpha\)) = 1000, parent population size (\(\mu\)) = 100, number of offspring (\(\lambda\)) = 50, archive size = 50, number of generations = 5000, crossover rate = 0.95, constant mutation probability = 0.01.

For comparing optimization with and without these new DoFs, three separate sets of experiments have been defined:

1. **Experiment Set (1):**
   Optimization without topology- and replication- DoFs. AQOSA supports general DoFs which are mentioned in Section 4.4.2, except topology DoF.

2. **Experiment Set (2):**
   Optimization with topology degree of freedom. AQOSA support all DoFs which are mentioned in Section 4.4.2 but not replication-DoF.

3. **Experiment Set (3):**
   Optimization with topology- and replication- degrees of freedom. AQOSA support all DoFs which are mentioned in Section 4.4.2 plus replication-DoF.

Each experiment set was run 30 times (90 experiments in total). For running experiment set (1), the baseline topology depicted in Figure 5.19 was used. On the other hand, for experiments sets (2) and (3), the AQOSA framework generates topologies by itself. In other words, they are not based on any baseline architecture.

Running on a powerful computer with a 12-core processor (each core runs 2.67GHz and 12MB cache) and 48GB memory, each experiment took between 1 to 2 hours. In total, the whole experiment took nearly 1 week of continuous computation.
Figure 6.11: Boxplot comparison of three experiments sets
6.4.2 Experiment Results

Figure 6.11 shows the differences between optimization with and without new DoFs. It shows the boxplot chart of 30 runs for each experiment set. It demonstrates the achievement of best solution in different objectives. In all of four graphs, lower values for objectives indicate better solutions because the optimization process was set to minimize all objectives values.

From these graphs, different effects for different objectives can be observed. For example, it shows us that for the processor utilization and cost objectives, the two new degrees of freedom significantly (approximately 65% and 75%, respectively) improve the results found by the optimization process: there are quite big improvement in these two objectives. For the objective safety, the two new DoFs slightly help in finding better solutions. And finally, for the objective bus utilization no improvement can be discovered as the results of using the two new degrees of freedom.

We expected that if the replication-DoF would have an effect, then this effect would mostly likely be large for the CPU utilization objective. Hence, to analyse the effectiveness of this DoF we compare the CPU utilization of experiment set (2) with that of experiment set (3). However, in Figure 6.11b both are far better than the results of experiment set (1). Figure 6.12 shows the boxplot chart of best CPU utilization in 30
runs only for experiment sets (2) and (3). Again, a lower value indicates a better CPU utilization. This means that running the algorithm with this DoF ended up with better solutions regarding the CPU utilization objective. Hence, the graph demonstrates notable improvement for the CPU utilization objective by using the replication-DoF.

To represent the coverage of all of five objectives, Figure 6.13 depicts a comparison of hypervolume indicators for three experiment sets. In this graph, higher values mean better coverage of the design space. We set reference points in hypervolume calculation to 2 (all values are between 0 and 1), because fix topology optimization concentrate on one region but optimization with new DoFs try to find solutions even in the boundary regions. So, reference points were defined a little far from boundaries. As can be seen in the graph, two experiment sets with the new DoFs perform considerably better than experiment set (1). Also, since the boxplot of experiment set (1) shows a smaller variance, it can be concluded that this approach finds solutions from a more limited area of the design space. Of course, experiment sets (2) and (3) depend, like any genetic algorithm, on sufficient randomness in the optimization process in order to achieve good coverage of the design space and hence in finding optimal solutions.

**Figure 6.13:** Boxplot comparison of the hypervolume of archive sets
6.5 Summary

This chapter studied the improvements that can be obtained by introducing new degrees of freedom in the architecture representation (RQ2). In this chapter two novel degrees of freedom were introduced: (1) a new method for varying the topology of system architectures, (2) allowing architecture to replicate software component instances. More degrees of freedom essentially enlarge the search space, and hence allow evolutionary algorithms to find better solutions. By running a very computationally-intensive experiment on an industrial case study, it was shown that optimization using this approach indeed finds better system architectures. These experiments bring empirical evidences that prove better solutions can be found by using these new degrees of freedom. Moreover the approach was still computationally feasible.

This method also opens the doors for the next research question in the next chapter: We would like to introduce knowledge-directed search. To this end, we envision that we first diagnosing bottlenecks in an architecture design and then apply suitable architecture tactics as transformation of this bottleneck of the architecture which removes or reduces this bottleneck. Topology variation is fundamental for this method because it is not possible to apply many common tactics without having the topology DoF in the software architecture. Moreover, using the DoF introduced in this chapter it is possible to develop a load balancing search operator and apply it in a smarter targeted manner. This approach could be very useful in design problems with hard deadline such as embedded systems.