A SOFTWARE ARCHITECTURE MODEL FOR SELF ADAPTIVE SYSTEMS USING PSO

Y.MOHANA ROOPAA, A.RAMA MOHAN REDDYB

1,2 Department of Computer Science and Engineering, 1 Research Scholar in JNTUA & Associate Processor of AITS, Tirupati. 2 Professor of SVU college of Engineering, Tirupati, India
Email:ymohanaroopa@gmail.com, ramamohansvu@yahoo.com

Abstract— Self-adaptive software systems modify themselves at run-time in order to control the satisfaction of their requirements under changing environmental conditions. Self-adaptive software system has been proposed as a good solution for run time changes. However, very few techniques are available to date for systematically building such kind of system. Aiming at this requirement, this paper presents a sound approach to derive a self-adaptive software architecture model. In this paper, we will propose PSO for developing self-adaptive software architecture based on the configurable components related to the application. Initially, the components are clustered based on an efficient clustering algorithm. After that self-adaptive architecture will be built by using PSO on the basis of clustering results.

Key words: Component, Software Architecture, Self-adaptive, PSO.

1. INTRODUCTION

Over the past decade, self-adaptation has increasingly become a fundamental concern in the engineering of software systems to reduce the high costs of software maintenance and evolution and to regulate the satisfaction of functional and extra-functional requirements under changing conditions. Even though adaptation mechanisms have been widely investigated in the engineering of dynamic software systems, their application to real problems is still limited due to a lack of methods for validation and verification of complex, adaptive, nonlinear applications. One major challenge in self-adaptive systems is to provide guarantees about the required runtime qualities. A self-adaptive system comprises two parts: the managed system that deals with the domain functionality and the managing system that monitors the managed system and adapts it to achieve particular quality objectives. The key underlying idea of self-adaptation is complexity management through separation of concerns. The construction of software in modern computing contexts is increasingly concerned with volatile, unpredictable requirements. There are limits to the designer’s capacity to predict and accommodate change. Research fields have thus emerged which seeks to increase the flexibility of systems, both before, but even after software has been built and deployed. Adaptation is achieved in Component-Based Design by leveraging the capacity of component technology to reuse and, perhaps more importantly replace, discrete functional components in a system. Such change can be achieved dynamically, even as a system runs, and range over minimal point algorithmic adjustment to wide ranging architectural modification. A final area of research in adaptation and software evolution is the use of middleware infrastructure to support architectural and functional change. Such approaches deliver programmable runtime support for system construction of restructuring that allows dynamic execution context to be accommodated. Dynamically adaptive systems capabilities include automotive systems, telecommunication systems, environmental monitoring, and power grid management systems. It needs to be able to tolerate a range of environmental conditions and contexts, but the exact nature of these contexts remains imperfectly understood. The need for adaptability is perhaps most acute at the “wireless edge” of the Internet, where mobile devices balance several conflicting and possibly cross-cutting concerns, including quality of service on wireless connections, changing security policies, and energy consumption. To meet the needs of emerging and future adaptive systems, numerous research efforts in the past several years have addressed ways to construct adaptable software. Despite these advances in mechanisms used to build decomposable software, the full potential of dynamically decomposable software systems can be realized only if the adaptation is performed in a disciplined manner. Software architecture design is considered of paramount importance to the software development life-cycle. It is used to represent and communicate the system structure and behavior to all of a system’s stakeholders. Additionally, architecture can facilitate stakeholders in understanding architecture design decisions and design rationale, further promoting a communication and understanding. Software architecture (SA) is considered of highest importance to the software development life-cycle. It is used to represent and communicate the system structure and behavior to all of its stakeholders with various concerns. Additionally, SA facilitates stakeholders in understanding design decisions and rationale, further promoting reuse and efficient evolution. One of the major issues in software systems development today is systematic SA restructuring to accommodate new requirements due to the new market opportunities, technologies, platforms and
frameworks. The ultimate goal of software engineering is to be able to automatically produce software systems based on their requirements. For the time being, we pass the synthesis of executable programs, and concentrate on the automated derivation of architectural designs of software systems. This is possible because architectural design largely means the application of known standard solutions in a combination that optimizes the quality properties of the software system.

II. RELATED WORK

Architecture-based management approaches promote the use of architectural models as guidelines for various management functions. Some of the recent researches done by the researchers are given in this section. Over the past decade the dynamic capabilities of self-adaptive software-intensive systems have proliferated and improved significantly. To advance the field of self-adaptive and self-managing systems further and to leverage the benefits of self-adaptation, there was a need to develop methods and tools to assess and possibly certify adaptation properties of self-adaptive systems, not only at design time but also, and especially, at run-time. Norha M. Villegas et al. Proposed a framework for evaluating quality-driven self-adaptive software systems. Their framework was based on a survey of self-adaptive system papers and a set of adaptation properties derived from control theory properties. They also established a mapping between those properties and software quality attributes. Thus, corresponding software quality metrics can then be used to assess adaptation properties. Software validation and verification (V&V) ensures that software products satisfy user requirements and meet their expected quality attributes throughout their lifecycle. While high levels of adaptation and autonomy provide new ways for software systems to operate in highly dynamic environments, developing certifiable V&V methods for guaranteeing the achievement of self-adaptive software goals is one of the major challenges facing the entire research field. Gabriel Tamura et al. Proposed a paper in which they have (i) analyzed fundamental challenges and concerns for the development of V&V methods and techniques that provide certifiable trust in self-adaptive and self-managing systems; and (ii) presented a proposal for including V&V operations explicitly in feedback loops for ensuring the achievement of software self-adaptation goals. Both of those contributions provide valuable starting points for V&V researchers to help advance this field. Self-adaptation has been widely recognized as an effective approach to deal with the increasing complexity and dynamicity of modern software systems. One major challenge in self-adaptive systems was to provide guarantees about the required runtime qualities, such as performance and reliability. Existing research employs formal methods either to provide guarantees about the design of self-adaptive systems, or to perform runtime analysis supporting adaptations for particular quality goals. Yet, work products of formalization were not exploited over different phases of the software life cycle. Danny Weyns proposed a paper, in which they have argued for an integrated formally founded approach to validate the required software qualities of self-adaptive systems. That approach integrated three activities: (1) model checking of the behavior of a self-adaptive system during design, (2) model-based testing of the concrete implementation during development, and (3) runtime diagnosis after system deployment. They have illustrated that approach with excerpts of an initial study and discuss for each activity research challenges ahead. Quality of software is one of the major issues in software intensive systems and it is important to analyze it as early as possible. An increasingly important quality attribute of complex software systems is adaptability. Software architecture for adaptive software systems should be flexible enough to allow components to change their behaviors depending upon the environmental and stakeholders’ changes and goals of the system. Evaluating adaptability at software architecture level to identify the weaknesses of the architecture and further to improve adaptability of the architecture are very important tasks for software architects today. Pentti Tarvainen proposed an Adaptability Evaluation Method (AEM) that defines, before system implementation, how adaptability requirements can be negotiated and mapped to the architecture, how they can be represented in architectural models, and how the architecture can be evaluated and analyzed in order to validate whether or not the requirements are met. AEM fills the gap from requirements engineering to evaluation and provides an approach for adaptability evaluation at the software architecture level. In that paper AEM was described and validated with a real-world wireless environment control system. Furthermore, adaptability aspects, role of quality attributes, and diversity of adaptability definitions at software architecture level are discussed. Over a period of some 20 years, different aspects of co-management (the sharing of power and responsibility between the government and local resource users) have come to the forefront. Fikret Berkes proposed a paper which focused on a selection of these: knowledge generation, bridging organizations, social learning, and the emergence of adaptive co-management. Co-management can be considered a knowledge partnership. Different levels of organization, from local to international, have comparative advantages in the generation and mobilization of knowledge acquired at different scales. Bridging organizations provide a forum for the interaction of these different kinds of knowledge, and the coordination of other tasks that enable co-operation: accessing resources, bringing together different actors, building trust, resolving conflict, and
networking. Social learning was one of those tasks, essential both for the co-operation of partners and an outcome of the co-operation of partners. It occurs most efficiently through joint problem solving and reflection within learning networks. Through successive rounds of learning and problem solving, learning networks can incorporate new knowledge to deal with problems at increasingly larger scales, with the result that maturing co-management arrangements become adaptive co-management in time.

III. PROPOSED METHOD FOR SOFTWARE ADAPTION MODEL USING PSO

Architecture models are an expression of the earliest design decisions and a means of abstraction understand the system. As the meanings of software architecture are many, the role of software architect has become very demanding. The level of abstraction has risen, required amount of cumulative knowledge has exploded and international and multicultural environments with geographically distributed development sites emphasize an ability to communicate ideas clearly.

3.1 Component Architecture For Software Adaption
If architecture is the set of design decisions, then documenting the architecture boils down to documenting the set of design decisions. This is generally not done, though. We can usually get at the result of the design decisions, the solutions chosen, but not at the reasoning behind them. Much of the rationale behind the solutions is usually lost forever, or resides only in the head of the few people associated with them, if they are still around. So the reasoning behind a solution is not explicitly captured. This is implicit knowledge, essential for the solution chosen, but not documented. The knowledge about software architecture and its environment is called Architectural Knowledge and has resulted in a paradigm shift in the software architecture community. The most important type of AK is architectural decisions, which shape software architecture. Other types of AK include concepts from architectural design, requirements engineering, people and the development process. Software architecture is generally the structure of components in a program or system, their interrelationships, and the principles and design guidelines that control the design and evolution in time. In Software Architecture Recovery the process integrates the existing architecture recovery tools to support architecture recovery process. The major drawback is that process automation, application of process and framework to large and complex software systems and refinement of process and framework based on experiences and integration with different development processes are not possible. The test cases we generated in our proposed method are cost and faults that can occur in software. Once the test cases are generated, the clustering process is carried out.

3.2 Clustering Of Test Case Using Birds Flocking Algorithm
The flocking algorithm defines a method for imitating the flocking activities of birds on a computer. Particularly, in flocking algorithm, there is no leader i.e., no global control. Here, each agent has direct access to the geometric representation of the scene. To discover the spatial data while searching for the clusters, flocking agents are changed into hunters with a foraging behavior. The flock always follows an exploring behavior, in which each and every member or agent first to discover the environment searching for goals whose positions are not known a priori, and then, after the goals are positioned, all the flock members should shift towards these goals. In birds flocking algorithm, the motion of flocks of birds is characterized as individual behaviors. It elaborates on a particle system. Each boid (agent) utilize an accurate geometric model for flight. The flocks normally steer toward the general heading of the rest of the flocks. The agents search the goals in parallel, and indicate the presence or dearth of significant patterns into the data to other flock members by varying the color. It responds only to the flock mates within a certain small radius. The flocking mates within a certain small radius. The flocking algorithm defines a method for imitating the flocking activities of birds on a computer. Particularly, in flocking algorithm, there is no leader i.e., no global control. Here, each agent has direct access to the geometric representation of the scene. To discover the spatial data while searching for the clusters, flocking agents are changed into hunters with a foraging behavior. The flock always follows an exploring behavior, in which each and every member

![Fig 1: Flow diagram of our proposed method](image-url)
applied in the opposite direction which is scaled by the inverse of the distance. Separation produces a force that steers the agent away from those in its neighborhood region. When separation applied to a group of agents they will spread out, trying to maximize their distance from other agents. That is a certain distance is maintained from other surrounding neighbor agents. Thus, crowding of agents is prevented by allowing them to scan a wide range of area.

3.2.2 Cohesion
In cohesion, a steering force is applied to move an agent to the center of mass of its nearby agents. This will make the group of flocks to move towards each other. The cohesion is calculated by finding all the agents in the confined neighborhood and computing the average of their position vectors. This gives the value, which is the center of mass of the neighbors, the region where the agent wants to reach. Once the center of mass is computed, the agent uses seek steering behavior to move to that location. In general, the cohesion gives an agent the ability to coalesce with other nearby agents (to form a group).

3.2.3 Alignment
Alignment drives the boids to head in the same direction with matching velocities. The average velocity of the flock mates in neighborhood is computed and they drive towards that velocity. This will make the group of boids to travel in the same direction at the same speed. Steering for alignment can be calculated by finding out all the nearby neighborhood agents and averaging their heading vectors. In general, the alignment gives an agent the potential to align with other nearby characters.

3.2.4 Flocking Algorithm for Clustering:
In this paper, a multi-agent adaptive algorithm called SPARROW algorithm is used for adaptive flocking, which identifies the clusters in parallel. It begins with a fixed number of agents that take up a randomly generated position. Then, a core point is identified as each agent moves around the spatial data testing the neighbor of each location. The neighbors of the identified core point are given a temporary label. These labels are updated as multiple clusters. Contiguous points belonging to the same cluster take the label corresponding to the smallest label in the group of contiguous points. The movements of the agents are all described in Reynolds’s model. The color is used as a communication device between the flock agents to indicate the roadmap they need to follow. The roadmap is adaptively attuned as the agents alter their color moving to explore data until they reach the goal. Consider a $d$ dimension search space in which the flocks move. Assume, N is the number of test cases and $t_i$ is the set of all the test cases \[ \{t_{1}, t_{2}, \ldots, t_{N} \} \]. Then, each agent is classified into above categories. The agents are clustered into groups by the birds flocking algorithm as shown in the fig 2.

![Fig 2: Test case Clustering using Birds flocking algorithm](image)

The following are the key features of our model, which is different from Reynolds’s:

- Alignment and cohesion do not consider blue boids, since they don’t move in a very attractive zone.
- Cohesion is the resultant of the heading towards the average position of the green flock mates (centroid), of the attraction towards reds, and of the repulsion from whites.
- A separation distance is maintained from all the agents, without considering their color. Normally, a flock is a group of agents all staying close to each other, and the cohesion component is responsible for this action. Each agent watch the position of other agents to observe if it is within a specified neighbor radius, that is, it checks to see which other agents are close enough to be considered flock mates. The positions of the eligible neighbors are averaged and the agent moves towards that position. In this way, each one aims to move towards the center of the flock that results in, all of them are staying close together. Agents will move towards the desired destination with a speed depending on their color. The green agents travel more slowly than the blue agents since they will discover denser zones of clusters. An agent will increase the speed to depart a vacant or dull region but it slows down to explore an interesting region more carefully. The variable speed has established an adaptive behavior in the algorithm. The movement and speed are changed by the agents based on their earlier experience represented from red and white agents. The cohesion component is computed by averaging the position of the all neighbors within the radius. In the merging stage, two diverse cases are handled: when having never visited points in the circular neighborhood and when having points belonging to diverse clusters. In the first case, the points are labeled and a new cluster is formed, whereas in the second case, all the points will be pooled into the same cluster i.e., they will get the label of the cluster discovered first. A cage effect is occurred during the simulations, that is, some agents are detained inside the regions bounded by red or white agents and would have no way to depart, wasting some valuable resources for the exploration. Thus, to shun this effect,
a limit is prescribed for their life. Hence, when their age goes beyond a determined value (max Life) they have been destroyed and regenerated in a new randomly selected location of the space. We need to utilize the flocking algorithm to discover the multidimensional space searching point. A continuous data point can be represented in a multidimensional Euclidean space, by simply normalizing its attributes. In the following, we have given a proper depiction of the extension of the flocking algorithm to multidimensional space. Consider a multidimensional space with \(d\) as dimension. Each boid \(k\) can be denoted as a point in the space having coordinates \(x_{k1}, x_{k2}, \ldots, x_{kd}\) and having directions \(\beta_{k1}, \beta_{k2}, \ldots, \beta_{kd}\), where \(\beta_{ki}\) gives the angle between the new direction of the boid \(k\) and axis \(i\).

Each boid will move according to the velocity \(v_k\) of the boid \(k\). For each iteration \(t\), the new position of the agent \(k\) is given by the following equation:

\[
x_k(t + 1) = x_k(t) + v_k \times c_{ki}, \quad \text{where } i=1\ldots d
\]

(1)

and \(c_{ki}\) represents the projection along the \(i\) axis of the direction of the boid \(k\). Each component is determined by adding the respective components of alignment, separation and cohesion.

i.e.,

\[
c_{ki} = c_{alignment} + c_{separation} + c_{cohesion}
\]

Thus, in a multidimensional space, the components are calculated as:

\[
c_{ki} = \prod_{j=1}^{d-1} \cos(\beta_{kj})
\]

\[
c_{ki} = \sin(\beta_{ki-1}) \prod_{j=i}^{d-1} \cos(\beta_{kj}) \quad i = 2\ldots d
\]

(2)

Once the clustering process using bird flocking algorithm is completed, the optimization of the clustered test cases is carried out using optimization algorithm. In our proposed method we use PSO as the optimization algorithm.

### 3.3 Test Case Optimization Using PSO

PSO originated from the simulation of social behavior of birds in a flock. In PSO, each particle flies in the search space with a velocity adjusted by its own flying memory and its companion’s flying experience. Each particle has its objective function value which is decided by a fitness function. PSO is an evolutionary computation technique which is very similar to that of the Genetic Algorithm where a particular system is initialized by a population of random solutions. In PSO along with each potential solution, randomized velocity is also assigned which constitute a particle. Each particle follows its coordinates in the problem space in connection with the best solution. Here the fitness value is also considered for the further process. This fitness value is referred to as \(pbest\). The location of these solutions is regarded as \(gbest\). In our proposed technique we have utilized a modified version of PSO. In this PSO we have assigned worst case as well along with the best case and also cross over operation is also included after the fitness selection which would further increase the possibility of selecting the best particle.

The PSO thus provides better solution and the steps involved are given in the below section.

#### 3.3.1 Steps in Particle Swarm Optimization:

The various steps involved for implementing the PSO is explained below,

i. First initialize a population of particles (solutions) with position and velocity chosen randomly for \(n\)-variable in the problem space.

ii. For each of these randomly generated particles evaluate the optimization fitness functions in \(n\)-variables.

iii. Now compare this fitness value with the particles \(pbest\) value. If these current fitness value is better than the \(pbest\) then chose the current fitness value as the \(pbest\) for the further processing.

iv. These fitness values is compared with the overall best previous values and if the current value is better than update the \(gbest\) for the current particles array index and value as the new \(gbest\).

v. Change the velocity and the position of the particle and then repeat the steps until the criterion of better fitness is obtained. The velocity and the position of the particle are varied with the help of the below equations,

\[
v_j(n+1) = v_j(n) + a_1 \cdot \left( p_j(n) - h_j(n) \right) + a_2 \cdot \left( g_j(n) - h_j(n) \right)
\]

(1)

\[
h_j(n+1) = h_j(n) + v_j(n+1)
\]

(2)

vi. The process is repeated until the solution with better fitness value is obtained.

In the above equations, \(a_1\) and \(a_2\) represents the acceleration constants that is needed for combining each particle with the \(pbest\) and \(gbest\). Updating the best position of the particle can be given as per the equation below,

\[
g_j(n+1) = \begin{cases} g_j(n), & \text{if } h_j(n+1) \geq k(g_j(n)) \\ h_j(n+1), & \text{if } h_j(n+1) < k(g_j(n)) \end{cases}
\]

(3)

The particle velocity at each dimensions are limited to the interval \([\pm V_{max}]\) are then measured and is compared with the \(V_{max}\). The \(V_{max}\) is an important parameter. The \(V_{max}\) helps in determining the resolution with which the region between present position and the target position are searched.
Depending on the $V_{max}$ values the particles are considered to provide a better solution. The above equations are utilized to calculate fitness of the solution and to select the better solution based on these fitness values. Hence PSO can be utilized to solve the optimization problems just like other evolutionary algorithms. It can be applied in various fields like signal processing, robotics, simulations applications etc. Here we have employed PSO in order to select the better solution for the test case optimization. The various test cases are processed in the algorithm to obtain the best result suited for software architecture.

IV. RESULTS AND DISCUSSION

In our experiment, we have utilized the Particle swarm optimization for the software architecture. The implementation is done in the JAVA platform and the results are given as follows. In our proposed method we have utilized the hospital management system as the source software. Based on the fitness value of the parameter chosen the whole process labors in Particle Swarm optimization. For the additional processing the parameter with high fitness value is chosen. For both Particle Swarm optimization and Genetic Algorithm the fitness values of the chromosomes are computed for dissimilar iteration and the results are charted. By seeing the Table I the fitness value for the suggested technique verified to be superior to the technique where GA is applied.

<table>
<thead>
<tr>
<th>Iterations</th>
<th>Fitness value</th>
<th>Proposed method using PSO</th>
<th>Using GA</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>15</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>13</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>13</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>12</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>12</td>
<td>13</td>
<td></td>
</tr>
</tbody>
</table>

Table I: Fitness value for Adaptive GA and GA for each iteration.

![Fig 3: Comparison of Fitness value between PSO and conventional GA](image)

![Fig 4: Comparison of Computational Time for PSO and GA](image)

Based on the above table, the graphs are plotted and from the graph shown in fig 4, it is clear that the fitness values of AGA converges quickly when compared to GA. The computational time is well thought-out as the most important issue in software architecture as made cleared in the preceding section. The computational time for the software architecture plan based on the chosen test cases by means of both the PSO and GA are after that computed and the resulting values are charted. The significances of the computation time we attained for different test cases are shown in Table II.

<table>
<thead>
<tr>
<th>Optimal Test Cases</th>
<th>Computational time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PSO</td>
</tr>
<tr>
<td>5</td>
<td>804</td>
</tr>
<tr>
<td>10</td>
<td>1122</td>
</tr>
<tr>
<td>15</td>
<td>1781</td>
</tr>
<tr>
<td>20</td>
<td>2078</td>
</tr>
<tr>
<td>25</td>
<td>2603</td>
</tr>
</tbody>
</table>

Table II: Fitness value for Adaptive GA and GA for each iteration.

The graph is designed for computational time for dissimilar test cases based on the values shown in above table. Using PSO and GA the graphical representation of computational time for our suggested method was shown in Fig. 5. As shown in the graph, the computational time for PSO has been less significant when match up to that of GA.

CONCLUSION

In this paper, we have proposed a system to design a component based software architecture using optimization algorithm like PSO. We have utilized the clustering process, bird flocking algorithm in order to cluster the test cases. The test suite optimization will
be done by employing Particle Swarm Optimization Algorithm (PSO). The result shows that our proposed method has delivered better results in terms of test case optimization values when compared to other optimization techniques like genetic algorithm etc

REFERENCES


